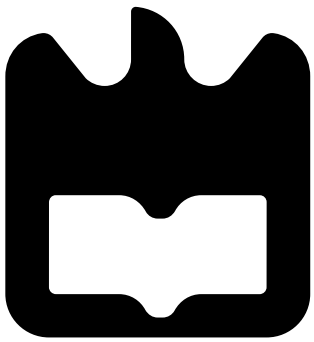
Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2014

Tiago Filipe da Graça Saraiva

Sintetizadores de Frequência Programáveis por USB

USB Programmable Frequency Synthesizers



Tiago Filipe da Graça Saraiva

Sintetizadores de Frequência Programáveis por USB

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requesitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica de Armando Rocha, Professor do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri / the jury

presidente / president

Professor Doutor João Nuno Pimentel da Silva Matos
Professor Associado, Universidade de Aveiro

vogais / examiners committee

Professor Doutor Armando Carlos Domingues da Rocha
Professor Auxiliar, Universidade de Aveiro (orientador)

Doutor Luis Manuel Santos Rocha Cupido
Director Executivo, Lc Technologies (arguente principal)

agradecimentos / acknowledgements

Gostava de aproveitar para agradecer a todos os que me apoiaram neste projecto e providenciaram valiosas críticas e orientação.

Os meus estimados agradecimentos, em especial, ao professor Armando Rocha pela disponibilidade que sempre demonstrou e pela ajuda preciosa que me foi dando ao longo da escrita e do desenvolvimento do projecto.

À Universidade de Aveiro por ter providenciado as instalações e infraestrutura para o meu percurso académico.

Quero também agradecer à minha mãe Maria Eugénia Graça por todas as oportunidades que me deu ao longo da vida e do meu percurso académico e profissional.

Finalmente, mas não menos importante, à minha noiva, Catarina Amaral, pela infinita paciência e fins-de-semana perdidos, e por todo o apoio que me deu incondicionalmente.

Obrigado.
Tiago Saraiva.

Resumo

Os sintetizadores em arquitectura Phase-Locked Loop (PLL) são actualmente a forma mais largamente utilizada de produzir sinais com frequências dentro de um intervalo a partir de um oscilador de frequência fixa.

Apesar disto, existem poucas opções no mercado de sintetizadores de frequência programáveis para aplicações de testes em campo ou amadores, sendo que as poucas opções que existem apresentam custos proibitivos.

Esta tese descreve os estados de design e produção de uma plataforma de baixo custo e facilmente customizável de sintetizadores de frequência em PLL programáveis por USB para os mercados amadores, de testes em campo e de laboratório.

Abstract

Phase-Locked Loop frequency synthesizers are the most widely used method of generating ranges of frequencies from a single fixed-frequency oscillator.
Despite this, there are few options on the market of frequency synthesizers for field-testing or hobbyist applications, and the ones that exist are quite expensive.
This paper focuses on the design and production of an easily modifiable, low cost platform for USB programmable frequency synthesizers for field testing, laboratory and hobbyist applications.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Purpose

The objectives of this thesis are to detail the designing, building, programming and usage phases of a Universal Serial Bus (USB) compatible controllable/programmable frequency synthesis platform.

Easy to use, good quality, USB controllable frequency synthesizers are, at the time of writing this paper, very hard to find off-the-shelf, or indeed quite expensive for all their intended applications.

There are mainly three starting objectives for the device designed in this paper:

1. That it is less expensive to produce than the off-the-shelf synthesizers already available;
2. That its output signal quality is suited for use in laboratory, field and hobbyist applications;
3. That it is easy to use. This includes being easy and fast to program and deploy without using external hardware or software;
4. That it can easily be tailored to work with different frequency ranges by changing only a few components;
5. That it provides an output power of about 7 dBm to be used as a local oscillator for double balanced ring diode mixers.

1.1.1 Some Practical Applications

One of the main objectives is that the devices described in this thesis are to be useful for several applications. They shall be able to be used to demodulate signals in laboratories for testing, where it is useful to adjust the frequency, even if slightly, or even sweep through various frequency settings at a given interval. They should also be usable in the field, programmed with a single fixed frequency, or an array of frequencies that can be cycled, without the use of a companion computer for programming needs. The low cost objective opens it to the hobbyist market, that is, people who would have a use for it in their own personal projects.

The chosen communication or programming standard is Universal Serial Bus (USB), since it is now included in most personal computers and doesn't require external hardware or software.

1.1.2 Off-The-Shelf Synthesizers

The market offering for USB controllable or programmable frequency synthesizers is somewhat scarce. The few available devices are of a high cost, especially for the hobbyist market. At the time of writing, a company called Quonset Microwave offers on their online website [1], an array of USB stick synthesizers. There are several down-sides with their offering. The main down-side is their price, ranging from 500 USDollars for a stick synthesizer, with a bandwidth of 1380 MHz to 4400 MHz, to 2650 USDollars for a 10 000 MHz to 20 000 MHz synthesizer. Another major down-side is that they can only be used with a PC, or at least a device with a powered USB port present. The main benefit of this offering is the low phase noise advertised, even in high bandwidth devices.

The company Windfreak Technologies, LLC., sells via their website [2], a set of RF synthesizers which are advertised as low-cost. This company offers an interesting solution by the name of "SynthUSBII" which is a portable, USB controlled signal generator with high bandwidth and non volatile on board memory. This device is in some ways similar to the devices this project focuses on creating but, even though it is the most affordable device that this company sells, it's still sold for 250 USDollars at present. There is also a drawback in the fact that this company uses a proprietary firmware to control their devices which isn't public. This means that it cannot be adapted for all practical applications, as the user needs to use the provided software for programming the device. This company offers other devices with different frequency ranges at prices as high as 1200 USDollars for a device which ranges from 50 MHz to 6000 MHz.

There are several other websites (e.g.: [3]) that offer kits for users to build their own synthesizers at home. This has the upside of being less expensive to build, but has the obvious downside of requiring that the user assembles the entire kit. They can be used without a USB connection present but the output frequency is impossible to be adjusted in this mode (it always uses the last frequency programmed in this mode). There isn't a wide array of documentation present in the website that presents final results, but it is safe to assume that the phase noise figures are not as good as the more expensive options.

The general drawback with most devices sold in the open market is that they operate at frequency ranges that are very high. This may not be advisable for all practical applications. We would like to develop a device that can be tailored to do this, but can also be adapted to present a narrow output bandwidth, which almost always brings the benefit of very good phase noise performance. Often, the devices available use switched frequency range synthesizers that do not, in most cases, have excelling performances.

1.2 Frequency Synthesizers

Electromagnetic signals are often transmitted using a carrier signal, which means that the input signal is modulated using an higher frequency. This allows certain benefits, among which is the possibility for several signals to share the same transmission medium.

Furthermore, electromagnetic waves at different frequencies have distinct propagation properties. For instance, radio waves transmitted from an antenna to another, may be propagated along the surface of the earth or through the atmosphere. Low frequency ground waves may be transmitted for several thousands of kilometres. However, at high frequencies, the energy losses experienced by the physical process of electromagnetic wave transmission only allow for propagation of ground waves for a few hundred kilometres [4].

While there are many kinds of modulation, nearly all of them aim to transform a signal from baseband to an higher frequency that may be more easily transmitted in whichever form is needed. Digital signals are modulated in order to make it possible to transmit the digital data through a physically transmittable wave, while analog modulation serves to change the signal at baseband into a signal that may be propagated at a more suitable frequencies.

As an example, in the case of Amplitude Modulation (AM), the amplitude of the carrier

signal is affected by the amplitude of the modulating signal, while in Frequency Modulation (FM), it's the frequency of the carrier signal that is affected by the original signal's amplitude value.

All modulation methods have in common the transformation of a signal into one with a different frequency. This is generally done with a frequency mixer, which works by creating two frequencies from two different input signals. The functions of a frequency mixer are explained in more detail in section 3.1.1. The modulating or demodulating frequency is often provided by a local oscillator. This may be a crystal oscillator, or some other kind of fixed-frequency oscillator. However, in order to change the frequency of the modulating or demodulating signal, which is to say that there is a change in the carrier frequency of the original signal, a replacement of the crystal, or LO in general, is required. We solve this by using a frequency synthesizer.

Frequency synthesis, then, is a method of generating one or several frequencies from one (or in some cases multiple) input frequency sources through the use of electrical or electronic components in a way that may be controllable as needed for the application.

Although there are several methods of deriving a range of frequencies from a single fixed-frequency oscillator, usually, the operation of a synthesizer is based on one of two methods: 'Direct' and 'Indirect'.

In the 'Direct' frequency synthesis method, the output waveform is generated using the reference input as a time base. There are both analog and digital versions of direct frequency synthesizers.

In Direct-Analog synthesis (also called mix/filter), the input signal may be divided or multiplied with other reference oscillators. The 'Direct' name stems from the fact that feedback loops are avoided. The output frequencies are isolated by means of filters. This synthesis method is seldom used nowadays due to its large setbacks. Even though phase noise performance can be very good due to the output signal being directly derived from the input, it can lend to very complex and expensive circuitry.

Direct-Digital synthesizers (which we elaborate on in section 3.1.2), use the reference oscillator signal as a time base for the generation of an arbitrary waveform in the output. The waveform of the output signal is digitally stored and then translated, by means of shift registers, to the output, at a rate which defines the resulting frequency in respect to the input.

The second method, indirect, uses a feedback loop in order to stabilize the output frequency.

The detailed workings of indirect or Phase-Locked Loop (PLL) synthesizers are explored in section 3.2 but in broad terms, the output waveform is generated via an oscillator which is controlled by a signal correlating to a difference in the phase of two input signals. If one of the input signals is a divided version of a reference signal and the other is a divided version of the output signal, the output frequency can be controlled by simply adjusting the division rates. The basic block diagram of a synthesizer of this type can be seen in figure 1.1.

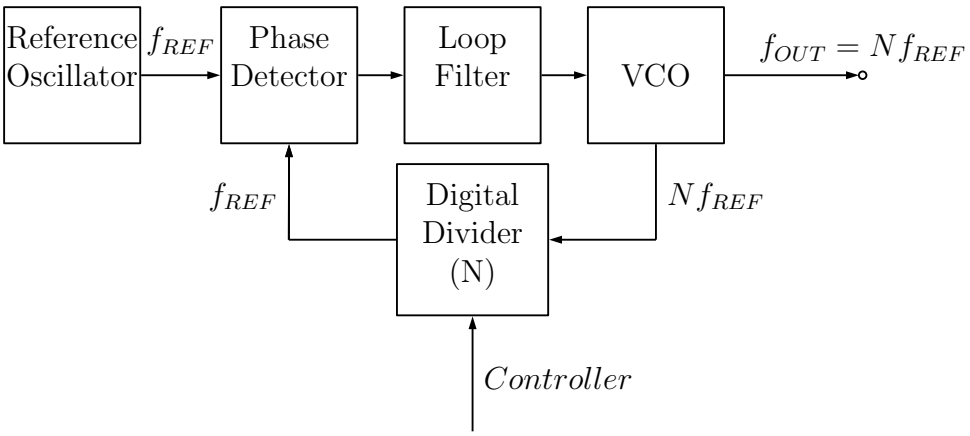


Figure 1.1: Basic block diagram of an indirect digital synthesizer

As can be seen in figure 1.2, in indirect-analog synthesizers, a mixer is used between the output of the Voltage Controlled Oscillator (VCO) and the input of the phase comparator. The VCO output is then with an external offset signal. By definition, when the loop is phase-locked, both the inputs of the phase comparator are at the same frequency (and phase). This means that the output of the VCO will be automatically adjusted until the output frequency is the addition or subtraction (depending on the filter) of both the reference and the external frequencies.

Frequency synthesizers are an integral part of modern communication systems, because they allow us to generate clock and oscillator signals of different frequencies from a local oscillator. They are used in radio receivers, mobile phone systems, satellite and GPS, among others.

Some applications in the field testing or laboratory environments require precise but fairly customizable frequency synthesizers. As seen above, the options in the market for this type of product are quite limited in number, and the few options that exist are often very expensive for private users. The purpose of this thesis is to create a platform that allows the user to easily

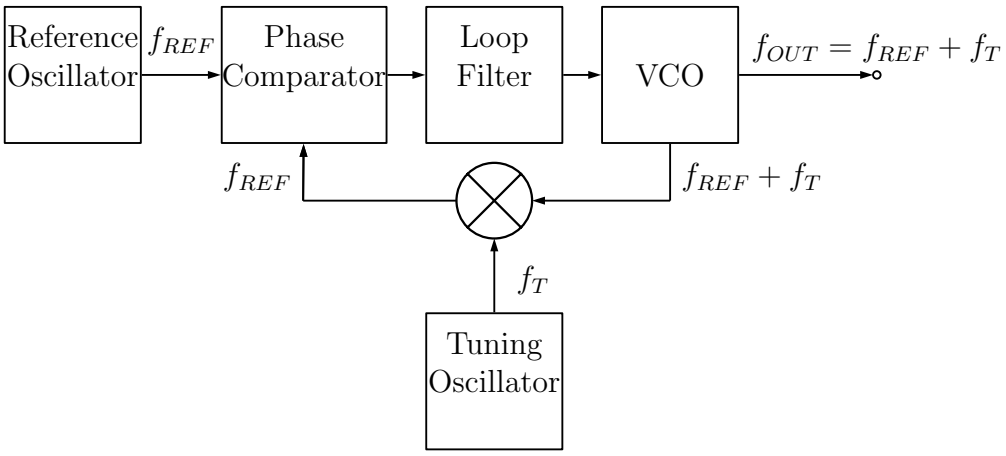


Figure 1.2: Basic block diagram of an indirect analog synthesizer

synthesize the required frequencies for his application with a substantially smaller price tag. The focus then, was to create a frequency synthesizer that was simple to program and easy to modify if needed. This is done by using off-the-shelf components that a normal user can replace and providing a comprehensive platform that can help the user modify the device without the requirement of writing new code or modifying the schematics or PCB of the device.

1.3 Structure

This paper deals with the creation of a Phase Locked Loop based frequency synthesizer, controlled by a micro-controller which is programmable via a USB connection with a PC, as well as all the software involved in programming and running the hardware.

The PC software was developed in the C++ language using the QT platform which makes it easier to develop for different platforms (Windows, Linux, Android), and the micro-controller software was developed in the C language using Microchip's MPLAB-X Integrated Development Environment (IDE). Both software use external libraries to handle Universal Serial Bus (USB) and the micro-controller software uses internal libraries for the Serial Peripheral Interface (SPI) functions.

In chapter 2 we explain some basic concepts of frequency synthesis phase noise as well as some of the basic parameters of Local Oscillators. Chapter 3 offers a more detailed description of different frequency synthesizer designs and compares different kinds of synthesis. In chapter

4, the hardware requirements for the device designed for the paper are detailed as well as the selection process of the different components that went into the construction of the test device. Chapter 5 details the software design process from the identified requirements to final development. Chapter 6 explains the process of designing and building the hardware itself. In chapter 7 the final product is analysed and in chapter 8 we make a critical analysis of the present work, results, and some suggestions on future work that could be done to improve the platform.

Chapter 2

Frequency Synthesis Concepts

2.1 Oscillators and Oscillator Specifications

Oscillators aim to produce a fixed-frequency signal as an output from a DC voltage input. These devices are essential in modern electronics as they effectively convert DC voltage to an AC signal.

The basic functioning of an oscillator can commonly be modelled as an amplifier inside a frequency dependant feedback loop. Figure 2.1 shows this basic model of an oscillator, where $A(j\omega)$ is the amplifier's gain and $\beta(j\omega)$ the feedback filter.

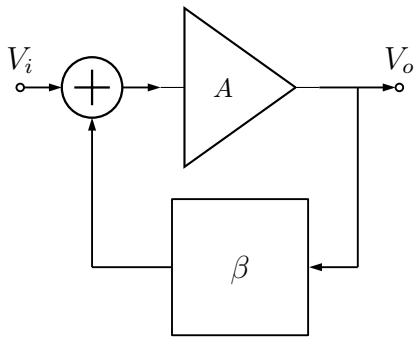


Figure 2.1: Basic model of fixed-frequency oscillator

The general expression of this oscillator model is:

$$\frac{V_o}{V_i} = \frac{A}{1 - \beta A}$$

The system will oscillate when $\beta A = 1$. This means that, as stated in [5], "at the frequency of oscillation, the total phase shift around the loop must be 360 degrees, and the magnitude of the open loop gain must be unity".

The filter used as the feedback loop determines the type of the oscillator. An RC oscillator has a filter consisting of resistors and capacitors, while LC oscillators use tank circuits consisting of inductors and capacitors. We discuss these further in section 2.1.2.

2.1.1 Oscillator specifications

Spurious and Harmonics

Spurious and harmonic components are signals present in an oscillator's output that are not wanted. Harmonics refer to multiples of the output frequency of the oscillator while spurs have no apparent connection to the oscillator itself. These types of components may be caused by numerous sources like reference frequencies in the signal source, power line or general interference and mixer products. Spurious signals are often measured in dBc (decibels relative to carrier frequency).

Pushing

Pushing refers to the phenomenon of the sensitivity of an oscillator's output frequency to its power supply voltage level. This is often expressed in Hz/V. If an oscillator is specially susceptible to pushing, this means that when there is an alteration of its power supply voltage, there will also be a change in the output frequency. The change in frequency can be either positive or negative.

As an example, Mini-Circuit's ROS-70-219+ oscillator has a frequency range of 50 MHz to 70 MHz and a typical pushing specification of 0.4 MHz/V, which is advertised as a low pushing number. If we assume that the oscillator is tuned to 50 MHz and there is a change of one volt in the supply voltage, the outcome would be a frequency of 50.4 MHz which is an increase of

approximately 0.8% in the output frequency.

Pulling

Pulling is the change in frequency due to the impedance of the load. In other words, oscillators that are susceptible to pulling experience an change in output frequency when there is an alteration of the load impedance.

Pulling is usually specified in data-sheets at a load return loss of 12 dB. The same Mini-Circuits ROS-70-219+ mentioned before has a pulling figure of 0.01 MHz at 12 dBr (dB return loss). This means that, because the system is matched to 50 Ω, there will be a drop of 10 kHz if the load changes to 100 Ω.

2.1.2 Examples of simple oscillators

Colpitts

The Colpitts oscillator was invented by Edwin Colpitts and is a type of LC oscillator design. In the Colpitts oscillator, which is depicted in figure 2.2, the feedback source is a capacitor voltage divider. The two filter capacitors and the inductor form a tuned tank circuit. The transistor in the common base configuration in this model together with the current source acts as the signal amplifier.

When power is provided, the C_1 and C_2 capacitors charge up and then discharge through the inductor L. The oscillations of the capacitors are applied to the emitter of the transistor and appear amplified at the collector output.

The oscillation frequency, that is, the frequency at which the phase shift across the loop provides the amount of positive feedback needed for sustained, un-dampened oscillations, is dependent on the values of the capacitors. Specifically, the ratio between the capacitors in series.

In practical terms, the oscillation frequency is determined by the resonant frequency of the tank circuit (resonant or tuned circuit composed of an inductance - L - and a capacitor - C). The resonant frequency of an LC tank circuit can be calculated with the expression:

$$f_r = \frac{1}{2\pi\sqrt{LC_T}}$$

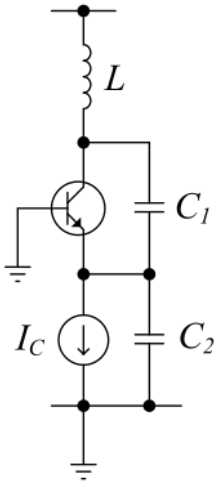


Figure 2.2: Common base Colpitts oscillator from [6]

Where C_T is the total capacitance of both capacitors connected in series: $\frac{1}{C_T} = \frac{1}{C_1} + \frac{1}{C_2}$ which means $C_T = \frac{C_1 C_2}{C_1 + C_2}$.

Hartley

Hartley oscillators are similar to the Colpitts design, however, the feedback source is now commonly an inductive divider (provided usually by a center-tapped inductor). As before, the resonant frequency of an LC tank circuit can be calculated with the expression:

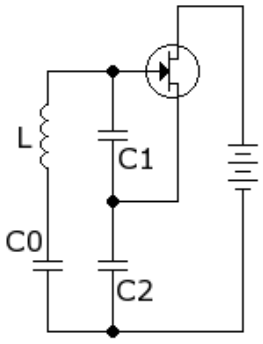
$$f_r = \frac{1}{2\pi\sqrt{L_T C}}$$

Where L_T in this case is given by $L_T = L_1 + L_2 + 2M$, that is, the sum of both inductances (the center tapped inductance acts as two inductances connected in series) plus the mutual inductance.

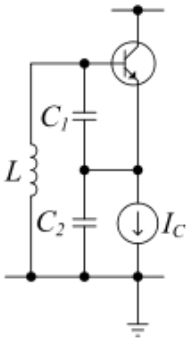
The resonating frequency is dependent on the tapping point of the inductance, as a fraction of the output signal is passed back to the emitter of the amplifier transistor. Because the phase shift between the transistor's emitter (feedback input) and it's collector (signal output) is 0, the feedback is positive, and the oscillating frequency is determined by the resonating frequency of the tank circuit.

Clapp

Clapp oscillators are also LC oscillators that use positive feedback to generate an output signal of fixed frequency. In reality, and as can be seen in figure 2.3, a Clapp oscillator is similar to a common collector Colpitts oscillator with an added capacitor in series with the inductance.



(a) Clapp Oscillator from [7]



(b) Common Collector Colpitts Oscillator from [8]

Figure 2.3: Comparison Between Common Collector Colpitts Oscillator and Clapp Oscillator

While in a Colpitts oscillator the oscillating frequency is changed by varying the capacitance of one of the feedback capacitors, in the Clapp oscillator, it is changed with capacitor C_0 (the capacitor in series with the inductance).

In the case of the Clapp oscillator, the total capacitance is given by $\frac{1}{C_t} = \frac{1}{C_0} + \frac{1}{C_2} + \frac{1}{C_2}$, which means that the resonant frequency is given by

$$f_r = \frac{1}{2\pi}\sqrt{\frac{1}{L}(\frac{1}{C_0} + \frac{1}{C_1} + \frac{1}{C_2})}$$

2.1.3 Voltage Controlled Oscillators

Voltage Controlled Oscillators (VCO) are used to generate variable frequency signals. These circuits output a signal which has a frequency (mostly) proportional to an input signal's DC voltage level. Some VCOs present a monotonic tuning characteristic which makes it that the output frequency has a single possible value across all tuning voltages.

There are two main roles for a VCO in a PLL synthesizer. The output frequency is used not only to provide an usable output for the device but also to be fed back into the Phase Detector (PD) through a frequency divider. This feedback loop allows for the PLL to generate a more stable frequency (i.e. more resistant to pushing and ambient electromagnetic noise) then is possible from a single frequency source.

While there are several types of voltage controlled oscillators, they commonly consist of a gain device (transistor or OPAMP) followed by a filter in a positive feedback loop. A buffer at the output helps reduce the effects of pulling by isolating the VCO from the load.

Frequency range

Frequency range is the minimum and maximum frequencies that can be obtained using a voltage controlled oscillator. The oscillator used as example in the previous sections (Mini-Circuits ROS-70-219+) works linearly between 50 MHz to 70 MHz.

In figure 2.4 we can see that the output frequency can be considered almost linear in the interval between 50 MHz to 70 MHz with tuning voltage between approximately 1 V and 4 V.

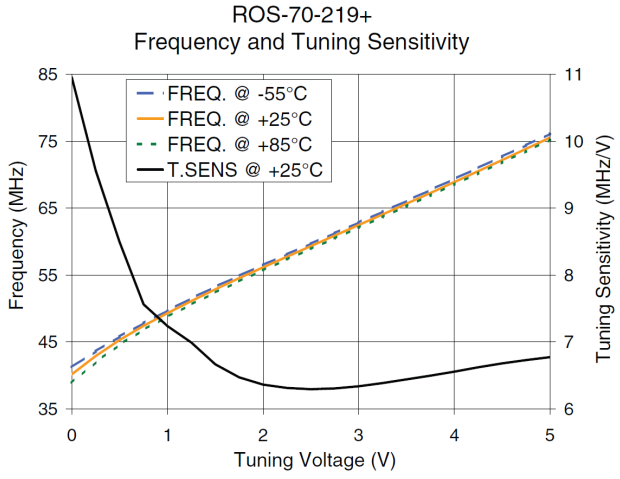


Figure 2.4: Frequency and tuning sensitivity of a ROS-70-219+ VCO from [9]

Tuning Sensitivity

Tuning sensitivity refers to the change in frequency at the output of the oscillator when input voltage is altered. This is commonly expressed as frequency change per unit voltage change (typically MHz/V).

Using as an example the same oscillator as before (Mini-Circuits ROS-70-219+), the sensitivity is specified as 7MHz/V (typ.) and the tuning voltage range is 0.5 V to 5 V Figure 2.4 shows that the tuning sensitivity actually varies wildly with tuning voltage, although it is more linear after approximately 1.5 V - maintaining a value between 6 MHz/V and 7 MHz/V.

Phase Noise

An ideal oscillator is a device that generates a single frequency waveform signal. In the case of controlled oscillators this frequency can be changed, but the output remains a single frequency figure at a given time. In reality, oscillators behave more like filtered noise generators.

As we will see in section 2.2, the output of an oscillator in the frequency domain has as distribution within an area around the target output frequency. An oscillator's phase noise is likely bandwidth limited and related to the Q factor of the resonator circuit. In a filter, the Q factor describes the filter's selectivity [10]. A high Q relates to a higher selectivity (smaller bandwidth relative to centre frequency) and lower insertion loss. In oscillators, a high Q factor means lower phase noise due to narrower bandwidth around the intended output frequency. The low-pass transfer function of an LC resonator, can be given by $L(\omega_m) = \frac{1}{j(2Q_L\frac{\omega_m}{\omega_0})}$ where Q_L is the loaded Q factor (the Q factor when a load is present), ω_m is the offset frequency and ω_0 is the centre frequency. [11]

As can be seen from the equation above, the bandwidth of the output signal of a resonator will be narrower with an higher Q factor.

Phase noise in an oscillator has two major types of components, the first are the discrete spurious signals which appear as distinct components in a spectrum analyser. The second component is the random phase fluctuations. This can be seen in image 2.2. The main sources of phase noise in an oscillator are thermal noise and flicker or 1/f noise.

Flicker noise refers to a form of noise which is inversely proportional to operating frequency. It is therefore most predominant in the spectral regions closer to the carrier frequency as we

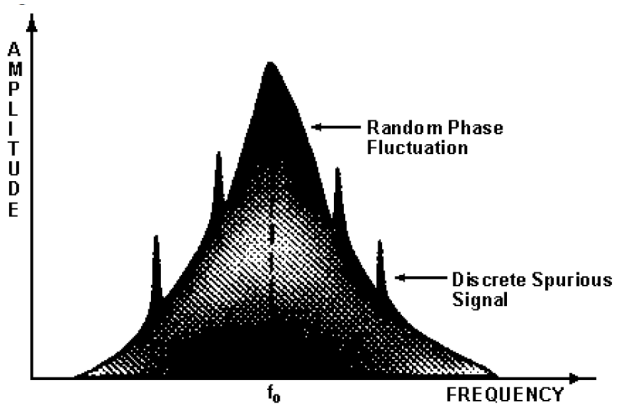


Figure 2.5: Representation of Phase Noise in a spectrum analyser [12]

---

can see in image 2.6.

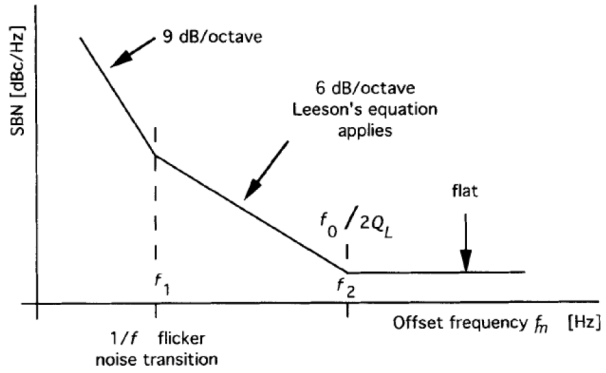


Figure 2.6: Representation of spectral regions of phase noise [13]

2.2 Details on Phase Noise

2.2.1 Introduction

A perfect oscillator would be one with the output function $V_o = V(t)cos(\omega_0 t)$. This would mean that it could be perfectly mathematically described by a sinusoidal waveform.

In reality, oscillators also produce noise, be it amplitude noise or phase noise. In real

oscillators then, a more accurate representation would be the output function

$$V_0 = V[1 + n(t)]cos(\omega_0 t + \Theta_n(t))$$

where $n(t)$ and $\Theta_n(t)$ are commonly zero-mean random processes, describing deviations of amplitude and phase respectively.

However, adding to the fact that amplitude noise is generally less noticeable than phase noise, it is generally less important in the design of emitter or receiver circuits. There is no way to decrease phase noise except by improving the oscillator's performance. Furthermore, phase noise causes limitations in both signal modulating, and receiving devices. [14]

The power spectral density of an oscillator's signal is ideally composed of a single line, meaning that the power output is concentrated in a single output frequency. Phase noise makes it so that power is randomly distributed in bands around the oscillator frequency such that the spectrum no longer looks like a single line. This means that phase noise can be measured using a spectrum analyser. However, this approach is conditioned by the phase noise levels present in the analyser's local oscillators. Moreover, analog spectrum analysers perform averaging, and there may be differences in frequency resolutions.

The major contributors to phase noise in oscillators are f^{-1} noise or flicker noise, thermal FM noise, flicker phase noise and the thermal noise floor.

2.2.2 Phase Noise in Free Running Oscillators

As previously discussed, if we ignore amplitude noise in the oscillator output, it can be mathematically represented by the expression $V_0 = Vcos(\omega_0 t + \Theta_n(t))$ where V is the amplitude and $\Theta_n(t)$ a random process representing the phase noise. Because phase noise is theoretically a zero-average random process we can model this behaviour with a sinusoidal function. This makes it easier to study the effects of phase noise.

As is done in [15], we will consider that $\Theta_n(t) = \frac{\Delta f}{f_m} sin(\omega_m t)$, where f_m is the frequency of the sinusoidal wave representing the phase noise and $\frac{\Delta f}{f_m}$ is the peak phase deviation, leaving us with

$$V_0 = Vcos(\omega_0 t + \frac{\Delta f}{f_m} sin(\omega_m t)) \tag{2.1}$$

We can separate the terms of the cosine

$$V_0 = V[cos(\omega_0 t)cos(\frac{\Delta f}{f_m}sin(\omega_m t)) - sin(\omega_0 t)sin(\frac{\Delta f}{f_m}sin(\omega_m t))] \tag{2.2}$$

In cases where the peak phase deviation is much less than 1 ($\frac{\Delta f}{f_m} \ll 1$), as it is demonstrated in [15], we have

$$V_0 = V\left\{\cos(\omega_0 t) - \frac{\Delta f}{2f_m}[cos(\omega_0 + \omega_m)t] + \frac{\Delta f}{2f_m}[cos(\omega_0 - \omega_m)t]\right\} \tag{2.3}$$

This means that when phase noise is present in the output, there are two additional frequency components on each side of the carrier frequency (at the frequencies of $f_0 \pm f_m$) with an amplitude of $\frac{V\Delta f}{2f_m}$. We can now expand this result to understand how the power spectral density of an oscillator looks. This can be seen in figure 2.5, which represents the power spectral density of an oscillator output.

Single-sideband (SSB) phase noise is a measurement of the noise in the frequency domain as measured with a spectrum analyzer. It's measured by quantifying the power of the signal in a 1 Hz bandwidth to a single side of the carrier frequency. Because of this, the units used are dBc/Hz (where dBc means dB below carrier).

Figure 2.7 shows the single-sideband phase noise measurements of a Mini-Circuits ROS-70-219+ commercial oscillator. In this case, the SSB phase noise would be approximately 100 dBc/Hz at 3 kHz offset from carrier frequency.

As will be seen in section 2.2.3, an important fact about phase noise is that when an oscillator's signal is multiplied in the frequency domain, this increase is also translated to the SSB phase noise. This means that if the frequency is multiplied by a factor of ten, the phase noise in the resulting signal will increase 20 dB.

2.2.3 Phase Noise in PLL Synthesizers

Sources of Noise Within the Loop

In the PLL synthesizer, the phase detector generates high amounts of transient noise at its frequency of operation. The VCO is modulated according to this noise as well as the output

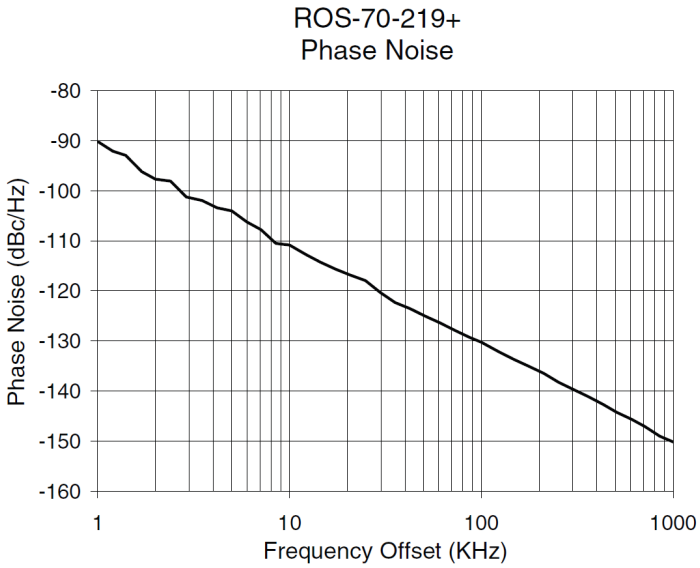


Figure 2.7: Phase noise analysis present in the datasheet for a commercial oscillator from [9]

control voltage of the Phase Detector (PD). This translates to spurious signals at offsets equal to the PD's working frequency. The loop filter is present at the output of the PD to minimize this noise. This filtering operation removes unwanted noise from the output but it also conditions the PLL's switching times, due to limiting the speed with which the tuning voltage changes at the VCO input, thus a narrower filter will reduce the PD's noise at the cost of the time required for the PLL to switch between frequency channels [16].

At frequencies much larger than the loop bandwidth, the major contributor to the overall phase noise is the VCO phase noise. However, at offset frequencies below loop bandwidth, the dominant sources of phase noise are reference noise or the synthesizer's noise floor, whichever is highest.

The phase noise introduced by the power supply, inductors or transformers, if applicable, is also to be considered when designing a PLL. Furthermore, good noise performance is more easily achieved if all components are correctly placed relative to each other in the PCB layout, so as to minimize the situations where a component more prone to create noise is placed next to one more sensitive to noise.

Effect of Frequency Division and Multiplication on Phase Noise

As we've previously discussed, a PLL works by dividing the output frequency of the VCO and comparing the resulting signal's phase with the reference oscillator. As such, it is imperative that phase noise after the division is as little as possible, because even a small amount of noise after division could prove troublesome when comparing phases with the reference. We have previously seen (in equation 2.1 for instance) that the instantaneous phase $\theta(t)$ of a signal with phase noise can be given by:

$$\theta(t) = \omega_d t + \frac{\Delta f}{f_m} sin(\omega_m t) \tag{2.4}$$

Thus, we can obtain instantaneous frequency (which is to say the frequency at a given time considering the phase noise) by deriving the phase over time, or:

$$\omega(t) = \frac{\partial \theta(t)}{\partial t} = \omega_d + \frac{\Delta f}{f_m} \omega_m cos(\omega_m t) \tag{2.5}$$

When dividing the frequency by N in a loop, the resulting signal will be:

$$\omega_0(t) = \frac{\omega_d}{N} + \frac{\Delta f}{N f_m} \omega_m cos(\omega_m t) \tag{2.6}$$

and the phase (which can be calculated by integrating the instantaneous frequency),

$$\theta(t) = \frac{\omega_d t}{N} + \frac{\Delta f}{N f_m} sin(\omega_m t) \tag{2.7}$$

Therefore, we conclude that dividing the frequency not only reduces the carrier frequency by N but also reduces the peak phase noise deviation by N without changing the phase noise's *frequency*. Multiplication has, obviously, the opposite effect.

Chapter 3

Frequency Synthesizer Techniques

3.1 Direct Synthesis

As briefly discussed in chapet 1, in frequency synthesizers, the term "direct" is applied to methods of generating frequencies which do not use feedback loops. Direct synthesis can be either analog or digital.

3.1.1 Direct Analog Synthesis

In an analog configuration of direct frequency synthesis, the output signal is derived from mixing signals of certain frequencies. When a frequency mixer is used to mix two signals at different frequencies (f_1 and f_2), the resulting output signal is comprised of two frequencies, one that is $f_1 - f_2$ and the other $f_1 + f_2$ as shown in figure 3.1.

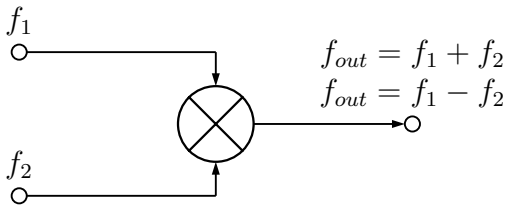


Figure 3.1: Basic block diagram of a frequency mixer

After selecting the desired frequency through a band-pass filter, we can use this technique as a basic building block for a direct analog synthesizer, allowing us to synthesize different frequen-

cies from two or more separate signals. As shown in figure 3.2, this process can theoretically be repeated as many times as necessary to reach the required frequency and resolution.

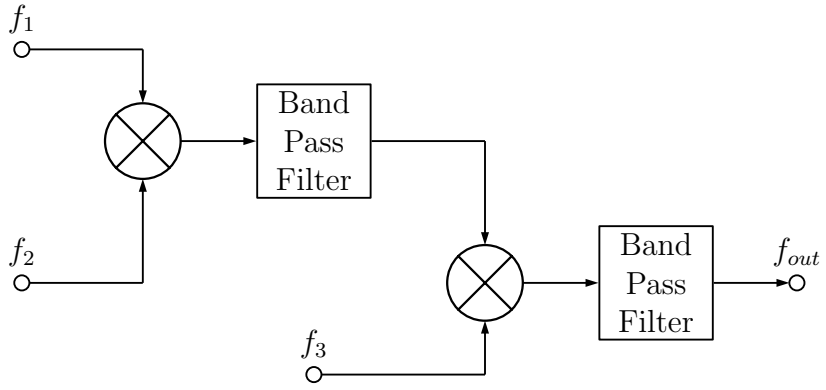


Figure 3.2: Basic block diagram of a direct analog frequency synthesizer

There are several disadvantages to this technique as the quantity of hardware components is very large. The quality (in terms of phase noise) of the output signal is heavily dependant on the input references, which would therefore need to be low-noise oscillators, resulting in a relatively high quantity of external components as well.

Furthermore, the mixers, filters and dividers are in the signal path, which contributes to output frequency phase noise.

3.1.2 Direct Digital Synthesis

Direct Digital Synthesis (DDS) is a type of frequency synthesis that works by generating arbitrary waveforms starting from a fixed frequency reference oscillator (typically an external crystal). A direct digital synthesizer is often composed by the reference frequency generator, a frequency or phase control register, a numerically controlled oscillator (NCO) or phase to amplitude converter, and a digital-to-analog converter (DAC). A basic block diagram can be seen in figure 3.3.

The phase accumulator register (PHA), comprised of an incrementer and a register, has its output incremented with every clock cycle of the reference frequency. When the maximum value is reached, the output cycles around to the starting value. The NCO produces a previously programmed output value based on the PHA's output. These values often comprise a full sinusoidal wave with each cycle of the accumulator. The output of the NCO is typically a

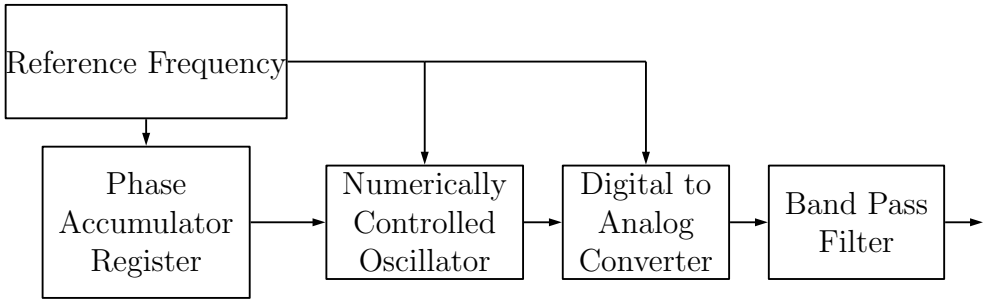


Figure 3.3: Basic block diagram of a DDS

digital (quantized and time discrete) form of the final output signal.

This waveform is then converted to an analog signal by the DAC. Because the DAC will usually contain spurious tones and harmonics, its output is then filtered by a low-pass or band-pass filter before the signal can be used. This will also eliminate the Nyquist frequency generated by the fact that this is a sampled process.

The frequency resolution in direct digital synthesizers can be arbitrarily fine, being that it is only limited by the size of the phase accumulator register, and the switching speed can be very high (depending at a minimum of the programming speed of the phase accumulator or NCO).

Due to the fact that the waveform is digitally chosen, different phase, frequency and amplitude modulations can be implemented using only small amounts of external software.

The output frequency of a DDS is calculated from the equation below.

$$f_{out} = M\frac{f_{ref}}{2^N}$$

where M (counter modulus) is the increment to the register on every step, and N is the bit size of the phase accumulator's register

According to [17], the minimum amount of reference-clock cycles that it takes to overflow the phase accumulator register is 2. This means that the maximum value for M is 2^{N-1}. If for instance we have a 4 bit register in the phase accumulator, and our modulus value is the maximum amount (an increment of value $2^3 = 8$), the output frequency will be $f_{out} = 8\frac{f_{ref}}{2^4} = \frac{f_{ref}}{2}$.

This means that this type of frequency synthesis may not always be the best solution for every use case, due to the fact that the reference frequency has to be at least twice as high as the maximum output frequency.

3.1.3 Frequency Multiplier

Frequency multipliers are devices used to synthesize signals with frequencies that are multiples of a lower frequency provided by a reference oscillator. These devices may be analog or digital but commonly non-linear devices (like transistors) are used. With these non-linear devices, the output signal is comprised of components at multiples of the input frequency (harmonics). This means that filtering is required to choose the correct harmonic, which means that the device is often followed by a band-pass filter. Figure 3.4 shows the schematics for a very simple diode bridge frequency doubler, with capacitors for filtering. This device has both negative and positive outputs.

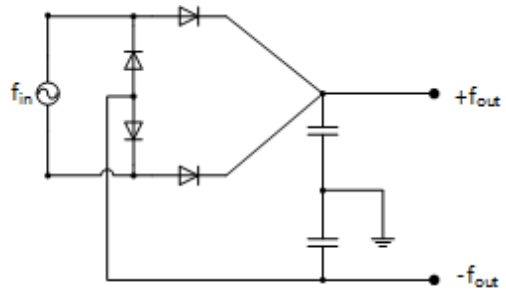


Figure 3.4: Full wave bridge doubler

3.2 Indirect Synthesis

Indirect synthesis refers to a method of synthesising frequencies where a feedback loop is applied between the output and input of the system that assures that the output frequency will stay within a certain threshold of the intended value.

One of the most widely used examples of indirect synthesis is the application of Phase-Locked Loops (PLLs). These are closed loop, frequency control systems.

A basic PLL system is essentially composed of a phase/frequency detector (PFD or PD for short), a loop filter - or integrator, a Voltage Controlled Oscillator and a frequency divider.

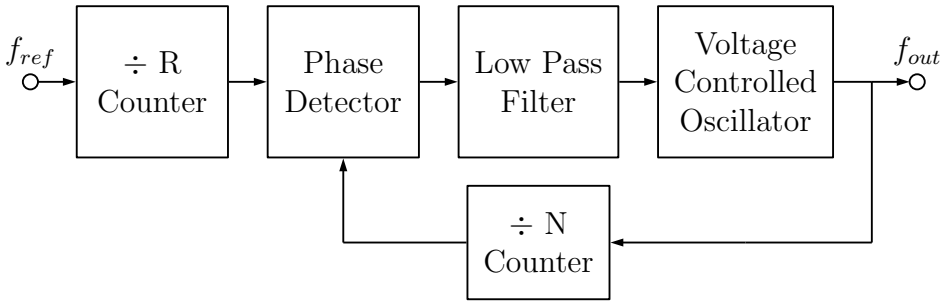


Figure 3.5: Simple block diagram of a basic PLL

Figure 3.5 shows a simple block diagram of a PLL. A frequency-divided reference signal is the first input of a PFD . The output signal of the synthesizer is the second input of the PFD, after passing through a frequency divider or counter.

The phase detector will generate current pulses at its output when there is a difference in its two input signals. The output of the PFD is then input to a low pass filter (or integrator) which will generate a voltage. This assembly of PFD and integrator, then, generates an output voltage proportional to phase difference between the two input signals.

The resulting signal is then fed to the VCO. The function of the VCO is to generate an output frequency that is proportional to its input voltage.

The loop is said to be locked when both inputs of the PD are at (to some degree of error) the same frequency and phase. When there are differences in phase, the voltage level at the output of the loop filter changes, which in turn causes a change in the VCO output signal varying it's frequency.

3.2.1 Phase Locked Loops

Along with the defining characteristics of an oscillator in an open loop, and adding to them, the parameters that characterize a PLL are:

Frequency range, or Tuning bandwidth - The range of frequencies that the PLL is able to output.

Frequency resolution - The minimum frequency increment. As we will see, this can be dramatically different with two PLLs of different types (Integer-N and Fractional-N) even if every other characteristic is the same.

Phase noise - This is discussed in detail in chapter 2. It represents the amount of energy being generated outside the centre frequency of operation.

Spurious signals - Signals outside the intended frequency output represented in the frequency domain as a spectral line. These are included in the noise analysis due to not being related to the output signal. Some PLL parameters are specific to the closed-loop system:

Switching speed - The speed with which the PLL tunes to a different output frequency.

Loop bandwidth - Defined by the loop filter. The loop bandwidth is a major parameter in PLL design as it limits both switching speed and phase noise, thus a compromise has to be made to optimize the PLL for the intended application.

Phase Detector

A linear analysis of a PLL can be done when we consider the phase detector a perfect multiplier with inputs $v_1(t)$ and $v_2(t)$, as stated in [18]. The output of the PD is then $K_m v_1 v_2$, with K_m having dimension of V^{-1}.

Disregarding noise, we let $v_1(t)$ and $v_2(t)$ be perfect sinusoidal reference signals with time invariant phase:

$$v_1(t) = V_1 sin(\omega_1 t + \Theta_1) \tag{3.1}$$

and

$$v_2(t) = V_2 cos(\omega_2 t + \Theta_2) \tag{3.2}$$

The output of the multiplier will be

$$v_{PD}(t) = K_m v_1(t) v_2(t) = K_m V_1 sin(\omega_1 t + \Theta_1) V_2 cos(\omega_2 t + \Theta_2) \tag{3.3}$$

or

$$v_{PD}(t) = \frac{1}{2} K_m V_2 V_1 (sin(\omega_1 t + \Theta_1 + \omega_2 t + \Theta_2) + sin(\omega_1 t + \Theta_1 - \omega_2 t - \Theta_2)) \tag{3.4}$$

if we consider that the purpose of the PLL is to match the two input frequencies of the PD, we can let $\omega_1 = \omega_2$:

$$v_{PD}(t) = \frac{1}{2} K_m V_2 V_1 sin(2\omega_1 t + \Theta_1 + \Theta_2) + \frac{1}{2} K_m V_2 V_1 sin(\Theta_1 - \Theta_2)) \tag{3.5}$$

We therefore have a low frequency term and a term at twice the input frequency. The high frequency term is later filtered in the loop filter, which means that, if we define $K_{PD} = \frac{1}{2} K_m V_2 V_1$, the output of the phase detector is simplified to

$$v_{PD}(t) = K_{PD} sin(\Theta_1 - \Theta_2) \tag{3.6}$$

Linear analysis

Although PLLs are non-linear, it would be complex, and outside the scope of this text, to use non-linear systems' analysis tools. As explained in [18], linear models are applicable when phase error is small, which is to say when the loop is locked. An adequate method of analysing PLLs is the Laplace transform.

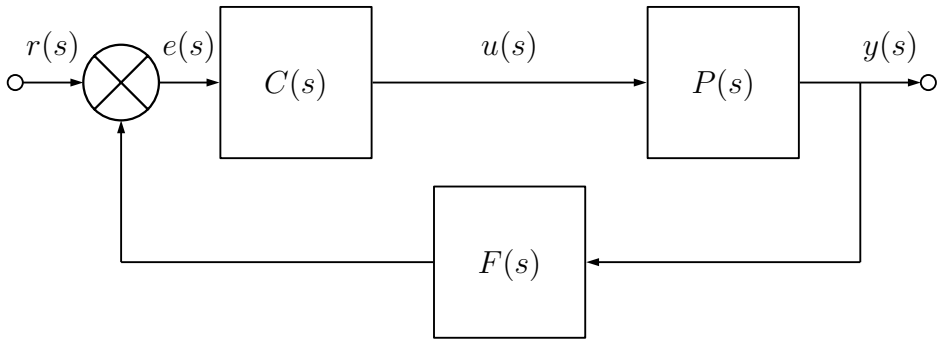


Figure 3.6: Basic negative feedback control system model

A basic control system is shown in figure 3.6 in which $r(s)$ is the reference signal, $e(s)$ the error signal ($e(s) = r(r) - [F(s) * y(s)]$) and $y(s)$ the output signal. $OL(s) = C(s)P(s)$ is the open loop expression of the system, while the closed loop expression is given by:

$$CL(s) = H(s) = \frac{OL(s)}{1 + OL(s)F(s)} = \frac{C(s)P(s)}{1 + C(s)P(s)F(s)} \tag{3.7}$$

In the case of a PLL, the open loop expression in the basic control system model may be expressed as in figure 3.7.

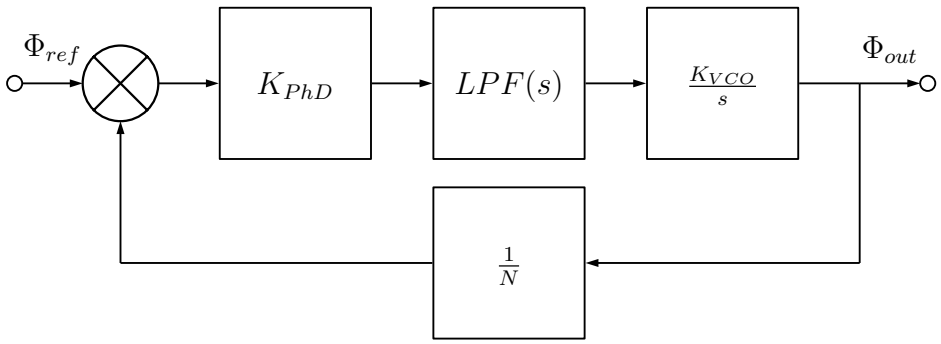


Figure 3.7: Basic linearised PLL model

As this linearised model shows, taking an input phase signal, the phase detector is represented as the subtraction of the two signals multiplied by a constant gain. This is because it produces an output voltage level proportional to the difference in phase between both signals. The loop filter is represented in the phase domain as a function and the VCO as gain and a signal integrator. N is the divider value. In this case, $\frac{1}{N}$ equals the $F(s)$ expression in the basic control system model, completing the feedback loop. This works so that when the loop is locked, $f_0 = Nf_i$.

We are then left with

$$OL(s) = K_{PhD}LPF(s)\frac{K_{VCO}}{s} \tag{3.8}$$

$$CL(s) = \frac{OL(s)}{1 + OL(s)F(s)} = \frac{K_{PhD}LF(s)\frac{K_{VCO}}{s}}{1 + \frac{K_{PhD}LF(s)\frac{K_{VCO}}{s}}{N}} \tag{3.9}$$

$$CL(s) = \frac{K_{PhD} LF(s) K_{VCO}}{s + \frac{K_{PhD} LF(s) K_{VCO}}{N}} \tag{3.10}$$

The number of poles at zero frequency in the open loop transfer function define the type of the PLL, while the number of poles in the loop indicate the order of the PLL. There will always be at least one integrator provided by the VCO so the simplest case for a PLL is type-I where the filter $LF(s) = K_{LF}$.

Type-I systems have the advantage of faster settling times but they suffer from non-zero steady-state phase error. This can be solved using a type-II PLL which can be achieved by using a simple low pass filter as the loop filter. $LF(s) = \frac{1}{1+sRC}$

$$CL(s) = \frac{K_{PhD} \frac{1}{1+sRC} K_{VCO}}{s + \frac{K_{PhD} \frac{1}{1+sRC} K_{VCO}}{N}} \tag{3.11}$$

$$CL(s) = \frac{N K_{PhD} K_{VCO}}{s^2 NRC + sN + K_{PhD} K_{VCO}} \tag{3.12}$$

and finally

$$CL(s) = \frac{K}{s^2 + s\frac{1}{RC} + \frac{K}{N}} \tag{3.13}$$

where $K = \frac{K_{PhD} K_{VCO}}{RC}$. This system can then be analysed using traditional control models for second order linear systems.

The basic second order control system is modelled by

$$CL(s) = \frac{A\omega_n^2}{s^2 + s2\zeta\omega_n + \omega_n^2} \tag{3.14}$$

We can then conclude that the system's natural frequency is given by

$$\omega_n = \sqrt{\frac{K_{PhD} K_{VCO}}{NRC}}$$

and the damping ratio given by

$$\zeta = \frac{1}{2}\sqrt{\frac{N}{RCK_{PhD} K_{VCO}}}$$

. To avoid excessive oscillation, damping ratios of second order systems are usually chosen as 0.7 which gives us $RCK_{PhD}K_{VCO} \approx \frac{1}{2N}$. Because $RC = \frac{1}{\omega_{LPF}}$, we have an easier job of designing the PLL synthesizer. Once we've chosen reference and output frequencies, only K_{PhD} and K_{VCO} are left to choose.

3.2.2 Integer N synthesizers

Traditionally, Integer-N synthesizers are the more widely used type of PLL synthesizers for local oscillators, according to [19].

As stated before, a PLL is comprised of several building blocks. These include the Phase-Frequency Detector (PFD), a loop filter, a VCO and a charge pump. These components are placed in a loop configuration that assures a stable output. There is technically a single input for a PLL which is the reference frequency. Disregarding for a moment the inner workings of the loop, the output of a PLL synthesizer is a sinusoidal signal with frequency $f_0 = Nf_{REF}$.

The N constant may be integer or fractional. In the case where N is integer, the synthesizer is said to be an integer-N synthesizer.

Figure 3.5 shows a basic integer-N PLL. We have two dividers in this case. One that operates over the input reference frequency and one that divides the output frequency.

The PLL system works by performing frequency multiplication in a negative feedback configuration. The reference frequency is often divided down by R before being fed to the phase detector (typically in the PLL's integrated circuit). When the phase detector is locked, its two input frequencies are the same. This means that $\frac{f_{REF}}{R} = \frac{f_0}{N}$, which leads to

$$f_0 = f_{REF}\frac{N}{R} \tag{3.15}$$

Thus, we can change the output frequency by simply changing the value of N. Of course, since N is always an integer, this circuit is limited in terms of frequency resolution. This is because with every increment in the value of N, the output frequency will be incremented by $\frac{f_{REF}}{R}$ which is, then, the minimum frequency resolution or minimum channel spacing. In the locked state, the frequency accuracy of the VCO is the same as the reference frequency.

As an example, if the reference frequency is 100 kHz and N is 1000 with an R value of 10, then the only output value that would lock the system would be 10 MHz. If the accuracy of the

reference frequency was 1 part-per-million (ppm), which means the reference would be accurate to $\pm 0.01\,\text{Hz}$, then the output of the PLL would be accurate to 1ppm or $\pm 10\,\text{Hz}$. Altering N to 1001 would mean an output frequency of 10.01 MHz.

While in the unlocked state, the phase detector generates an output proportional to the difference of its two input frequencies, which will then cause a change in the output frequency of the VCO, until the above equation is satisfied.

The popularity of integer-N synthesizers can be attributed to their simplicity. However, when relatively finer frequency resolutions are needed, the value of R would have to be higher or f_{REF} smaller. This would increase settling times. It would also require higher N division numbers, which could increase phase noise significantly, as seen in section 2.2.3.

3.2.3 Fractional N synthesizers

As discussed in the previous chapter, and as explained in [16], frequency multiplication significantly increases phase noise.

Thus, integer-N synthesizers have some limitations relating to phase noise performance. Since the frequency source is multiplied by $\frac{N}{R}$ to generate the output, increasing the value of N causes an increase in phase noise. An option to reduce the value of N would be to provide a higher frequency from the reference source, but this will limit channel spacing or frequency resolution (since this is dependent on the value of N).

In integer-N configurations then, a compromise has to be made between the switching speed and the phase noise and spurs generated by the device.

The solution comes in the form of Fractional-N synthesizers. These make it possible to achieve frequency resolutions that are fractions of the reference frequency. This is done by dynamically altering the value of N between N and $N+1$ during locked state. The ratio between the length of time the value of N stays at N and the length of time it stays at $N+1$ is, then, added as an arbitrary fraction to the value of the multiplication of frequency.

This allows for the frequency resolution to be smaller than the reference frequency, which in turn allows for a higher reference frequency and a lower value of N, thereby reducing noise.

The output function of a fractional-N synthesizer is then: $f_0 = \frac{f_{REF}}{R}(N + \frac{K}{F})$, where F is the fractional modulus of the circuit and K is the fractional channel of operation. F in this case

dictates the fractional resolution of the circuit. For instance, a value of $F = 5$ would indicate that the output resolution is $\frac{1}{5}$ of the reference.

Chapter 4

Hardware

4.1 Requirements

The project involved creating an easily programmable frequency synthesizer using the PLL method. Although a basic PLL synthesizer's design is not overly complex, creating a synthesizer for high frequencies with low phase noise and generally good signal characteristics is somewhat complicated using conventional methods and tools. In order to simplify hardware design, and increase reliability, it was deemed that a commercial PLL synthesizer should be used.

Commercial PLL synthesizers are often programmable by Serial Peripheral Interface (SPI), a standard that allows communication between a master (host) and multiple slave devices.

As discussed previously, in order for the device to be easily programmable, it would have to support the Universal Serial Bus (USB) standard. This standard is readily available in all recent personal computers and simplifies the user's process of programming different frequencies to the device. A micro-controller was chosen to perform this task.

4.1.1 Universal Serial Bus

Universal Serial Bus (USB) is a Plug-and-Play standard that aims to standardize the connection of peripherals to computers. It is now used for both transferring data and energy to devices, effectively replacing several types of serial and parallel interfaces.

USB was used in this project for two main reasons: Firstly, it is an easy standard to

implement seeing as though most micro-controller chips have hardware support for it out of the box; Secondly, the high availability of the standard in personal computers. This means that the synthesizer used in this project can be programmed from any recent computer that has USB ports as long as proper software is available.

4.1.2 Serial Peripheral Interface

Serial Peripheral Interface (SPI) is a serial data transmission standard. It is widely used in the researched PLL ICs for programming and data communication with the programming hardware. SPI buses operate in full duplex mode.

One of the devices operates in master mode, while a second or several other devices are operated in slave mode.

SPI buses commonly use four wires for communication. One of the lines operates as a clock, while two different lines transmit bits of information from the master to the slave and from the slave to the master. This communication is done simultaneously and synchronously with the clock line.

The frequency of the clock used for communication is flexible and is usually dictated by the slave's SPI module's maximum frequency of operation.

In this project, the chosen micro-controller communicates with the PLL through SPI for programming purposes. It is, then, imperative to select a micro-controller with support for this technology.

4.2 Hardware

4.2.1 Commercial Synthesizers

Nowadays, there are hundreds of different synthesizer IC families on the market. The characteristics of which vary wildly, as exemplified in table 4.1. The first step in choosing an adequate circuit is to define the desired characteristics. First of all, we define a frequency interval so we can focus on comparing different characteristics between similar hardware.

A medium-high frequency interval (around 2 GHz) was chosen to simplify the design of

further models of the hardware. This makes it so that if lower frequencies (around several hundred MHz) or higher frequencies (around 4 GHz or higher) are chosen at a later time, little to no changes will have to be made to the circuit schematics or the PCB design. The latter requirement also limits our choice of VCO. For instance, if choosing a VCO that works at around 2 GHz, similar pin-compatible VCOs that work at different frequencies will preferably be available from the same manufacturer.

One further requirement set to the synthesizer IC choice was that only Integer-N synthesizers would be chosen. Although in theory fractional-N synthesizers offer the advantages of improvement in phase-noise performance and faster lock times, their output is typically more susceptible to spurious content. Furthermore, they are usually more complex synthesizers to program. [19]

Company	Model	Freq. Range (MHz)	Type	Normalized Phase Noise (dBc/Hz)
Analog Devices	ADF4112	200 to 3000	Integer-N	-215
Analog Devices	ADF4151	500 to 3500	Both	-221
Texas Instruments	LMX2485	500 to 3000	Fractional-N	-206

Table 4.1: Table of some commercial synthesizers

The ADF4110 synthesizer family (which contains, among others, the ADF4112) includes devices able to produce output signals with frequencies ranging from 50 MHz, to 3.7 GHz. The pin compatible 4153 device offers a good fractional-N alternative and ranges from 500 MHz up to 4 GHz, while the 4106 chip produces frequencies up to 6 GHz. Replacing the ADF4112 with any of these devices is straightforward and requires no modifications of schematic or PCB. It will, however, be necessary to choose an appropriate VCO and realize the required changes to the filter.

Since this project aims to create a platform that can be easily used to develop new oscillators with different frequencies by simply replacing the VCO, the ADF4112 was the chosen synthesizer. This guarantees that by replacing the VCO for a pin compatible model that operates at different frequency intervals, and the loop filter components, we are able to create a new

synthesizer that generates different frequencies, without needing to develop a new design or PCB. The ADF4112 offers a middle ground, as it has relatively good phase noise performance, as well as a high operating frequency range.

Choosing this synthesizer also means that no changes need to be made to the micro-controller software for any of the other devices in the family. Both the computer client and the micro-controller software shall support the generation of frequencies from 50 MHz, to 3.7 GHz out of the box, being that the only modifications needed are to the physical device.

One other benefit of using a synthesizer that outputs frequencies at the approximate middle of the values possible for the project, is that, due to the particularities of high frequency PCB design, it is easier to design the original PCB layout for higher frequencies first as there are few adjustments that would be necessary to make the design work with smaller output frequency values.

While there is less need for adaptations for use of lower frequencies, higher frequency outputs have stricter requirements in terms of PCB layout, thus designing the PCB for the lowest possible output frequency given by the synthesizer IC family would mean rework was probably needed if a new 3 GHz device was to be assembled for instance. Designing the original PCB for frequencies closer to the top of the frequency range assures that there is minimal need of PCB redesign and rework when higher (or lower) frequency devices are needed.

4.2.2 Micro-controller

Because the chosen synthesizer is programmed via a Serial Peripheral Interface (SPI) compatible connection, there is a benefit from the micro-controller having SPI capabilities. Other important specifications would be USB capabilities and multiple power management modes - including sleep mode that allows the MC to be effectively shut down so as to minimize interference with other more sensitive hardware components. Furthermore, memory is needed to store programming words. Internal memory helps to simplify design and so that was added as a requirement for the choice of micro-controller.

Although there are many kinds of micro-controllers commercially available, a larger personal experience with Microchip PIC devices narrowed down the choice of MCU for the project.

PIC micro-controllers were originally created by General Instruments in 1975 as a Programmable Interface Controller (PIC) for the CP1600 CPU. In 1985, a new PIC version was

updated with EPROM in order to produce a programmable I/O controller. Microchip later registered the trademark PIC and changed the acronym's meaning to Programmable Intelligent Computer.

Nowadays there are several PIC families produced by Microchip Technology Inc. such as the 8-bit PIC10, PIC12, PIC16 and PIC18 families, the 16-bit PIC24, and the 32-bit PIC32 family.

The main differences between different architectures (aside from the obvious difference in the size of the data bus, register bus and instruction set given by the type of family - 8 bit to 32 bit), are the price and features. Traditionally, the 8 bit families are less expensive, which means they sport less memory space, less non-volatile memory and fewer peripheral support and features.

For small production numbers, 8 bit PIC MCUs have prices ranging from 0.50 € for the more basic models up to 14 € for the more complete models. 16 bit families have a range from around 3 € to approximately 10 €. The basic 32 bit models are somewhat more expensive (starting at around 4.5 €) while the most expensive models are in line with the most expensive 8 bit models.

There are several models in all architecture families that support all the features needed for the project. According to the Farnell on-line site (an electronic component supplier) there are around 60 PIC models that support both SPI and USB[20]. Microchip allows data words to be written in the program's flash memory. As only a few data words will need to be stored in non-volatile memory, EEPROM memory is not a necessity, and only a few bytes of free flash memory will suffice for the use of this project.

According to the Microchip website[21], in the 8-bit families, the PIC18 Architecture is the only architecture supporting USB and SPI. The 16-bit PIC24F family supports USB and SPI, with the added benefit of being designed for low power and low cost applications. The 32-bit micro-controllers are more complex and designed for more complicated applications.

Due to the fact that this project aims to be easily modifiable and upgradable, the PIC24F family was chosen. This family not only supports USB-OTG (USB device is able to work as a slave or a host) and SPI, but it also features an integrated graphics controller to implement graphical displays. This allows future projects to easily add new features to the device with little PCB modification. For instance saving frequency and lock-time measurements to log files

in external USB pen drives, or adding a LCD display to show the user pertinent information about the state of the device.

As such, the PIC24FJ32GB002 device was chosen. This is a 28 pin device with a 16-bit modified Harvard architecture and sports 32KB of Flash program memory. It has all the features described above, with a price in the middle of the range for the PIC MCU ICs (around 7€).

4.2.3 Voltage Controlled Oscillators

One of the objectives for the project was to create an easily modifiable platform that would be able to generate a wide array of frequencies by modifying the synthesizer chip and the VCO (along with the loop filter).

In order to allow the change of oscillator without a design modification being needed, to change the footprint on the PCB for instance, the chosen VCO would have to be fully compatible pin and supply voltage wise. The range of output frequencies allowed by the synthesizer family chosen is 50 to 3.7 GHz.

The Mini Circuits ROS line of synthesizers offers a multitude of chips that cover the entirety of the required spectrum. Table 4.2 shows a sample of the various components of this family that are compatible amongst themselves and their respective output ranges.

ROS-100+	50 to 100 MHz
ROS-200+	100 to 200 MHz
ROS-400+	200 to 380 MHz
ROS-540+	390 to 540 MHz
ROS-1300+	400 to 1300 MHz
ROS-2050-719+	1020 to 1980 MHz
ROS-2800-719+	1400 to 2800 MHz
ROS-3150+	2650 to 3150 MHz
ROS-3760+	3200 to 3760 MHz

Table 4.2: Some oscillators from Mini Circuit's ROS family of VCOs

As stated in section 4.2.1, a design choice was made to develop the device using a range of frequencies near the top of the frequency range for the synthesizer. The chosen VCO for the

project was the Mini Circuits ROS-2015+, with a range of 1975 to 2015 MHz. This device has a limited tuning dynamic range compared to some of the oscillators shown above, but has very good open loop phase noise performance as a result.

Chapter 5

Software

5.1 Micro-controller Software

5.1.1 Software Planning

The micro-controller's software was designed to do two tasks: Receive and store programming words from a USB port on the device's flash memory; Program these words to the synthesizer when the user so desires.

As previously discussed, the micro-controller chosen was a Microchip PIC24FJ32GB002. This micro-controller has several features that we desire for our project.

Based on the desired features, a software planning stage took place in which the software was detailed and implemented. The micro-controller software works in the manner depicted in figure 5.1. As can be seen, at power-on, the device executes initialization commands. These include, among others: preparing inputs and outputs, loading configuration words from Flash program memory to RAM and checking for USB connectivity.

The initialization function also programs the PLL with the last programming word programmed before the last shut down. This ensures that the device may be used with no USB connection to any host and start with the previous settings.

After this initialization stage, the software enters it's main activity cycle. This cycle consists mainly of checking for USB activity and performing pending operations.

If USB activity is detected, the software will read the USB port and perform the command

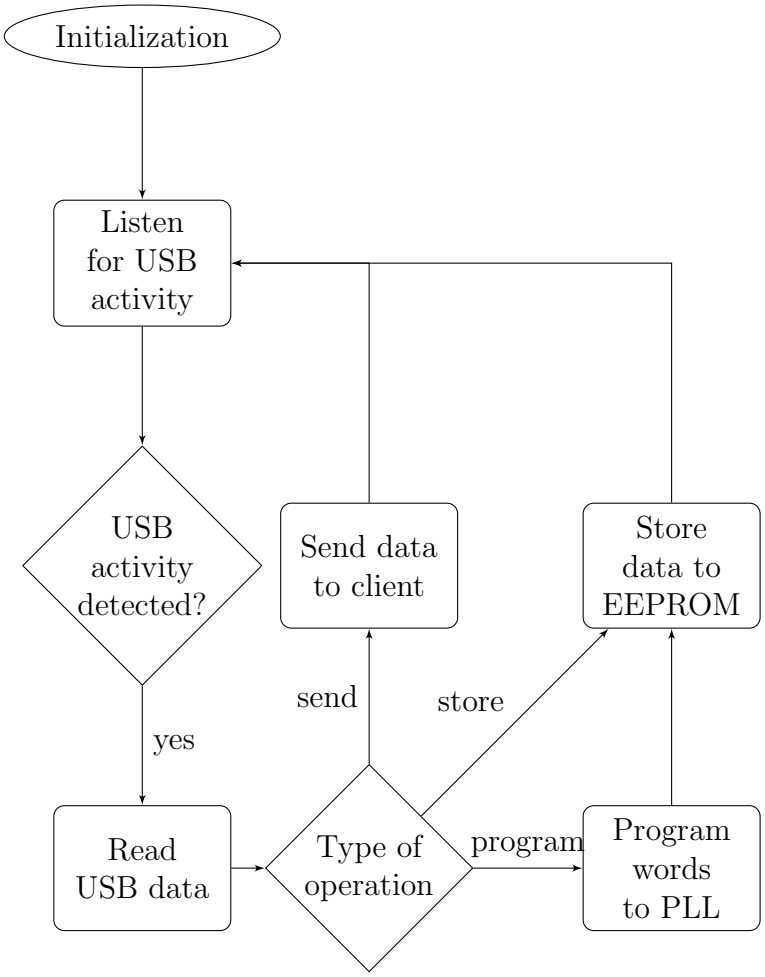


Figure 5.1: Basic diagram of micro-controller software

as sent by the PC client. This will cause an evaluation of the received data which is then processed according to what the user wants to do. Data may be saved to memory to be used later or immediately programmed to the PLL device.

This may entail programming the PLL direct and storing the data on the non-volatile memory of the micro-controller, or simply storing the words on the memory (without the programming step).

The Interrupt Service Routine (ISR) is called when an user input is toggled. This routine functions in the way depicted in figure 5.2.

The ISR is automatically called when one of the inputs is triggered. It makes a decision on

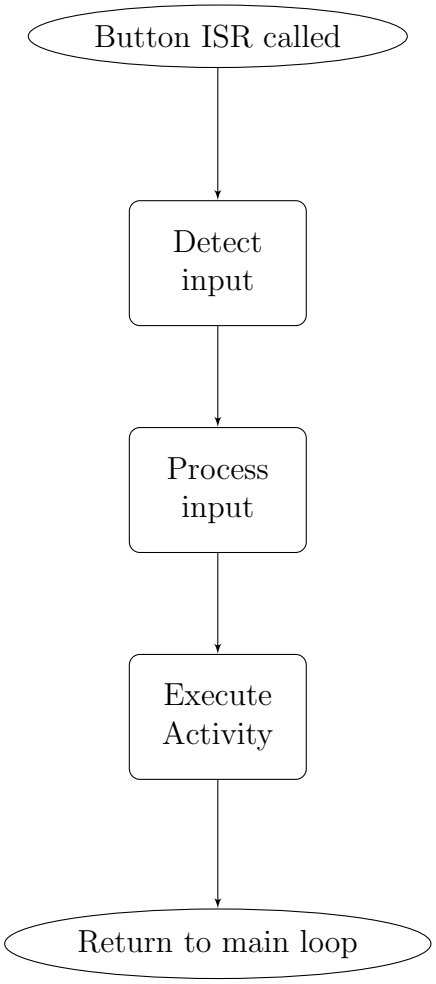


Figure 5.2: Basic diagram of ISR

the task to perform based on the input that was triggered. There are several types of different inputs that will be processed, however, the processing time of the input is kept to a minimum so as to minimize the effects of the program not executing the main loop for a large amount of time.

Three hardware push-buttons are available to the user, which can be programmed to perform different tasks. One of the buttons triggers the "Next Programming Word" function that will tell the micro-controller to program the PLL with the next word present in the RAM or Flash memory. A second button is programmed to switch the PLL to the previous word in the word list. This list works as a circular list. As a design choice, it supports 16 programming settings. This number was chosen because it was deemed sufficient for when the device is used away

from a PC which can program new words, This also simplifies the programming process as we will see in section 5.3.

One of the micro-controler's input pins connected to the PLL IC MUXOUT pin, this can be programmed in the programming word previously sent to the PLL to be triggered when there is a phase lock. It may also be set to a scaled version of either the reference frequency or the output. In this case, the micro-controller will read this input and decide what task to perform based on the last programmed word. For instance, if the last programmed word tells the PLL IC to output in this pin the Lock Detect (LD), a task will be set to upload the "Lock Achieved" signal to the USB host (if there is any).

There currently is no plan to implement a "Read N Divider Output" or "Read R Divider Output" functions from the PLL IC MUXOUT pin in the cases where the LD is not the programmed function. This can, however, be easily implemented in future versions of the micro-controller software. A useful application for this function is for testing the hardware and software when no spectrum analyzer is present, although a frequency divider would be necessary for the micro-controller to be used to analyze the output, which would subsequently entail a loss in detection precision.

5.1.2 Software Development

The micro-controller software for the PIC was developed using Microchip's own Integrated Development Environment (IDE) MPLAB X, and compiled using Microchip's XC16 compiler. The device can be programmed using the manufacturer's programming and debugging tools. In this case, the tool used was for programming was PICKit3.

The concept was done in a modular design and this was kept for the actual code written. For instance, the initialization function which is called at initial Power-On, will perform all needed initialization steps (for instance for the USB and SPI modules) have been taken by the initialize function and then program to the PLL the word map programmed in the previous session.

The USB functions used are provided by Microchip and do not allow for easy modularity. These had to be kept in the main cycle to avoid further complications and thus cannot be called directly by any other function. To keep the program simple, the CDC (Communication Device Class) code from Microchip was used to help with the communication between the device and

the PC. This device class, although fulfilling all the requirements and needs for this application, is somewhat limited. It works by emulating a serial console, reading data and replying as text. The protocol for textual communication between the devices is described in section 5.3.

To mitigate this fact, the main cycle was kept as simple as possible. The only operations performed in the main cycle are the USB Presence Checker, Activity Checker, and a Pending Task Verifier function.

The Presence Checker verifies that a USB cable is connected and supplying power to the USB presence pin. If there is no cable present then there is no activity to be checked, the cycle restarts. The Activity Checker checks the USB port for incoming data and stores any incoming data in a buffer to be processed in the next cycle. The Pending Task Verifier checks if there are any bytes ready to be processed. These bytes may instruct the software to perform actions and/or send results back to the PC Software. All other inputs (buttons) are on interrupt routines which means there is no need to check their value every cycle, they will be properly processed when their values change.

SPI Programming

As previously discussed, the chosen micro-controller supports a version of SPI communication out-of-the-box, using pre-configured pins for the SPI communication. However, the communication standard used in the ADF4110 family, while compatible with the Serial Peripheral Interface standard, is not quite equal.

The main difference between the communication standards in both devices is the presence of two separate data lines. While the actual SPI standard uses two lines for simultaneous transmission and reception of data (Master-Out/Slave-In and Master-In/Slave-Out), in addition to the clock and slave selection lines, the ADF4110 family chips only support one data line (Slave-In/Slave-Out), along with the clock line and a load enable line.

Although Microchip offers on their website well documented code examples and libraries for using SPI programming with their chips, the fact that there is a single data line is an impediment in and of itself of using this code. This is due to the fact that these example libraries are made for standard SPI applications.

There is another major setback in using Microchip's libraries, which is that it seems that sending words with sizes larger than the device's internal register size are not always correctly

supported. That is, if, like in out case, the device was a 16 bit micro-controller, the library only supported sending words through the use of the device's SPI port that were smaller or exactly 16 bits in size. The Slave Select line in Microchip's SPI implementations is invariably raised (it is an active low line) after the set amount of bits is sent. This is not ideal due to the fact that the chosen synthesizer, which is programmed using 24 bit words, loads the stored data into the latches when the Load Enable (the closest to a slave select line there is in the ADF4110 family implementation) line is raised.

After some attempts of trying to make the available code to function properly, it was deemed necessary to write an ADF4110 compatible SPI function. This function will write any value using only the three lines required by the ADF4110 family chips: the data input line, the clock line and the load enable line:

- Clock Line;
- Data Input Line: the polarity of this line is changed after every clock period to the logical value of the bit, starting with the most significant bit;
- Load Enable Line: starts high, goes to low before starting transmission in order to enable CMOS input in the chip and goes high once the entire word is transmitted.

5.2 PC Software

The platform chosen to develop the PC software was the QT Toolkit. This is cross-platform application framework, which allows us to develop the software for Windows and easily (with minimal changes) build it for other platforms (like Linux) later if desired. QT is widely used for developing GUI applications and has good support both for front-end development and back-end development. Programs using the QT Toolkit are developed using the standard C++ language and several libraries for an easier development of graphical applications. The compiler used for the development was Microsoft's MSVC (Microsoft Visual C++) compiler, and GDB (GNU Debugger) for debugging.

5.2.1 Desired Features

The PC software had to be able to send programming words to the MC and have them saved to memory. Other desired requirements would be to be simple to use and cross-platform.

The first version of the programming software involved calculating the programming words and sending them to the device. A calculator was then added that would create the words given the intended output frequency and the input reference frequency.

An important point to stress is that as the project was to develop an easily modifiable platform for creating different local oscillators that operate at different frequency intervals, the programming software would have to be customizable to the synthesizer in use.

It is hard to align both the ease of use and the ease of customizability at the same time. Different synthesizer - VCO combinations would require different programming words. As such, the approach of using a platform that ties in the output frequency or overall behaviour with a given set of words, isn't desirable.

Modifying the synthesizer - VCO combination (or even just the VCO, as the chosen synthesizer works for output frequencies of a large range) shouldn't entail heavy changes to the program's code. Therefore, a menu was included that helps adjust the calculated words with different synthesizers.

5.2.2 Software Development

USB Interface

An external library (QTSerialPort) was used for USB communication from the PC to the synthesizer. This library works as an external module for QT, making it easier to implement communication with any serial interface. As stated before, Microchip's CDC driver works by opening the USB communication as a simple serial transmission port on the PC. Therefore the QTSerialPort module can be used as is to transmit ASCII frames to and from the device.

The QtSerialPort module offers functions for opening and closing serial transmission, as well as reading and writing bytes to the line. A separate C++ class was then implemented to adapt to the device's software design. This entails wrapping the Open and Close functions in the device's protocol opening and closing functions, i.e., the module's "openCommunication" function should only result in an error free output if both the serial port was indeed opened and

the device responded to the "SYN" request with "ACK". Which means the opening procedure would fail if the device was not ready or the configured port was pointing to a different device.

The class that implements this was named Protocol class. This class works by providing two separate classes to implement messages passed between the devices.

- Generic_Message class - which is used for messages that do not require inputs or outputs (for example, SYN and FIN).
- Defined_Message class - which is used for messages that are used to send or receive specific data (SEN or REC respectively).

The Generic_Message class is comprised of a Byte Array object for the command and another for the expected response (IN and OUT), as well as methods to set and retrieve their respective values. The Defined_Message class extends the Generic_Message class and adds two byte arrays (DATA and RESPONSE) for data input and output respectively. Furthermore, the Protocol class implements all the methods for communication. Since the communication configuration used is master $\rightarrow$ slave, the reception of data from the micro-controller is always a consequence of a sent message.

Therefore, the "send" function of the Protocol class is implemented by two different methods, depending of the type of message to be sent. Both methods write data in steps and a failure to write one of the steps results in an error and the subsequent steps are not sent. The Defined_Message send method's structure is explained below in pseudo-code:

```
1  send (message)
2  {
3    if (connection_is_established)
4    {
5      write(start_byte)
6      if (success)
7      {
8        write(message.command)
9        if (success)
10       {
11         write(message.arguments)
12         if (success)
13         {
14           write(stop_byte)
15           if (success)
```

```
16            {
17              read(reply)
18              reply.response =
19                get_response(reply)
20              reply.acknowledgment =
21                get_acknowledgment(reply)
22
23              if (acknowledgment == reply_acknowledgment)
24              {
25                message.response =
26                  reply.response
27              }
28            }
29          }
30        }
31      }
32    }
33  }
```

If any of the steps fails, the function returns the appropriate error to the calling function and the message pointer's reply field is empty. Any errors returned by any of the methods of the Protocol class are sent to the main program loop where they are handled and passed on to the user if they are critical and cannot be automatically recovered. This way, the user always has the feedback of any successful or unsuccessful operation.

Programming Words

Because the micro-controller software is more difficult to modify or customize, and has to be more time and memory efficient, the decision was made to put the bulk of the processing in the PC client. This implicates more intricate PLL programming word validation and managing. The words may be customized in the PC and then sent directly to the synthesizer when they are needed. The synthesizer then needs only to either save the words to flash memory or program them directly to the PLL.

Between the different PLLs in the ADF4110 family, they support frequencies between 80 MHz and 3700 MHz. To simplify the development of the client, the decision was made to focus on the programming format of these synthesizers.

The configuration settings (figure 5.3) on the PC software, include a "Synthesizer Options" section, which may be modified to suit the required reference counter value for the board. These settings are automatically saved to an external file and are loaded at start up. This means that

if the user wants to switch between two boards with different PLLs or VCOs, all he will have to do is switch between configuration files. This allows different board to be fabricated with no changes to the PC client code.

As can be seen in the main client window (figure 5.4), words can be added by entering the desired output frequency and clicking the "Add to Map" button. This will calculate the counter words based on the reference counter, minimum frequency and maximum frequency configured in the settings.

This can only be achieved using a quite comprehensive structure for handling the programming words. The decision to keep the software compatible with only the ADF4110 family resulted in the added benefit of freeing up the time necessary for implementing cross-synthesizer compatibility, which resulted in more time to develop a more complete structure for the programming words of the synthesizer family

Every section of every programming word is defined as its own object which contains the value as an unsigned 32 bit integer bit mask as well as the index of the option in the customization window (which are in the same order as in the family data-sheet) when applicable.

As an example we will look at the Reference Counter Latch. This latch contains the 14 bit reference counter value as well as DLY/SYNC bits, a Lock Detect Precision bit and two Anti Backlash Width bits. Every one of these sections is implemented in its own class, each with an option index and value which relate to each other. Every class has implemented set and get methods for these values as well as bit mask setter and getter methods.

A parent class is then implemented that contains all other classes in a word. The sections are initialized from the bit mask of a 32 bit word when the parent class is initialized or from the index and values of its sections. Every component section of the word may be modified individually from this class. A $GetMask$ method is also present which returns the 32 bit unsigned integer that is then encoded for sending to the synthesizer.

When the "Add to Map" button is clicked, the expected output frequency entered in the "Frequency" box is evaluated. If the expected frequency falls within the limits defined in the configuration, the A and B counters are calculated according to the expression $f_{OUT} = [(P * B) + A]f_{REFIN}/R$, where P is the pre-scaler value and R is the reference counter value. Both of these values are defined in the configuration because they are a result of either the synthesizer used or the filter, meaning they change with the synthesizer used and the output

frequency range. Both the values of the A counter and the B counters are automatically calculated. As not every frequency may be achieved due to the phase detector frequency (which directly affects the step between two output frequencies), the calculated values are the ones which allow us to achieve the closest output frequency to the one entered by the user.

Once the calculations are performed, a new latch map position is initialized with pre-set default values for every other field. The A/B Counter is configured according to the values calculated and the new word set is added to the map and saved to a word file. The "Latch Map" is the name given to the file where the programming words are saved. This file is automatically loaded at start-up (if present) or initialized when a new word is added. The latch file is always synced with the word map in memory. Every position of the Latch Map consists of a name (which is set to the actual frequency achieved by the calculated parameters), an Initialization Latch Value, a Reference Counter Latch Value, an A/B Counter Latch Value, and a Function Latch Value.

Every single field of every value may be modified using the "Customize" button. Once again, all values are saved and synced to the Latch Map file whenever a value is modified. If either the A or B counter values are modified in the Customization window, the new name is calculated when the "Apply" button is clicked, and automatically substitutes the previous name, thus keeping the name in accordance with the calculated values. If the user wants to rename a Latch Map position, he may also do that and click apply without changing the A or B counter values.

5.3 Communication Protocol

The USB driver on the micro-controller works by checking the USB Serial IC for incoming or outgoing data and then performing the required action. The data is received in a stream of bytes, which means there has to be an agreed format for the information to be successfully transmitted between the USB host and slave.

Since transmission coding/decoding and error checks are already performed the USB ICs in both devices (PC and micro-controller) and in the drivers, the information is presented to the "Read USB Data" function as an array of characters (one byte each). The "Start Stream" and "End Stream" bytes are present as a check that the PC software is computing the information in the right way and that no random data is being received in the MCU. These are set to $7E$

Start Stream	Word 1	Word 2	Word 3	Word 4	End Stream
5 Bytes	6 Bytes	6 Bytes	6 Bytes	6 Bytes	1 Byte

Table 5.1: Example of the Communication Protocol Implementation

in hexadecimal and corresponds to the ~(tilde) in the ASCII table.

Table 5.1 specifies the protocol used to transmit data between the two devices. The start-end frame characters delimit the frame. The number of characters between them is a function of the command in the first three bytes. The programming words of the PLL used are 24 bits long (three bytes). For reasons explained below, these were split into half-bytes and encoded to values more easily handled by the used libraries, leaving 6 bytes for each programming word. Furthermore, there are four different latches for this device family.

Because of the limitations of using the simpler CDC libraries supplied by Microchip, the communication frames are sent in ASCII. Because of this, if for instance a byte was sent with the value of 0, which is perfectly common when transmitting programming words, the device or computer drivers may interpret this as a null character and not deliver it to the destination. The only valid characters of a frame for the implemented protocol are capital letters and numbers, and the tilde (~) character. The word bytes are parsed from their hexadecimal values to their respective representations using the numbers 0 through 9 and the capital letters A to F.

Example:

Command & Word map number	Init Latch Word	Reference Latch Word	A and B Counter Latch Word	Function Latch Word	End Stream Byte
~REC4	5FC093	4208A8	0266ED	5FC092	~

The word bytes or commands are then reconverted when received in the devices, shifted and OR'd to their correct positions in the word value and is processed. To keep the cycle interruption as short as possible and prevent the microchip software from becoming bloated, verification of the programming word is performed by the PC Software. This also prevents the need of re-writing the micro-controller's software and reprogramming the hardware board when a new PLL IC or VCO is replaced. This makes the software adhere more closely to the previously stipulated requirements.

Command	Argument(s)	Function
SYN	-	Initiates a connection
REC	Word Index and Word Array	Commands the synthesizer to receive this word array and store it in memory.
PRG	Word Index and Word Array	Commands the synthesizer to receive this word array, store it in memory and then program it to the PLL.
SEN	Word Index	Commands the synthesizer to send back the word stored in this position.
FIN	-	Terminates the connection

Table 5.2: Available Commands

5.3.1 Available Commands and Responses

Although communication is performed in a master → slave configuration (which means there is no way for the synthesizer to initiate a new communication to the PC), the REC command is not the only one available to transmit data to and from the synthesizer. Table 5.2 shows the available commands and their purposes.

When a command is sent from the PC to the synthesizer, a response is always sent. This informs the PC software that the command was successfully received and processed. Commands are always initiated by the user, so a response is useful for the PC to provide feedback to the user as a way to inform the user of what has happened. Table 5.3 explains the responses to the commands specified in table 5.2.

While the acknowledgement indicates the correct parsing of the command and respective data, the response arguments convey the successfulness of it's execution. As an example, in the PRG (Program) command, for instance, the PC software will send the following data:

Command & Word map number	Init Latch Word	Reference Latch Word	A and B Counter Latch Word	Function Latch Word	End Stream Byte
~PRG2	5FC093	4208A8	0266ED	5FC092	~

Command	Arguments(s)	Acknowledgement
SYN	-	ACK
REC	1 byte per word, indicating the successful storage of the word. (0 for success)	ACK
PRG	1 byte per word, indicating the successful storage of the word. (0 for success) + 1 byte indicating the successful programming of the PLL.	ACK
SEN	Word Array	ACK
FIN	-	BYE

Table 5.3: Command Responses

The response from the synthesizer will have the following format:

Start Byte	Init Latch Word Stored	Reference Latch Word Stored	A and B Counter Latch Word Stored	Function Latch Word Stored	Words Programmed to PLL	Ack. and End Byte
~	0	0	0	0	0	ACK~

The action result bytes are zero in case of success, but they change accordingly to the output of the functions used. The *WriteFlash* function returns 0xF in in case of error (for instance if the index passed for the word storage relates to a memory position outside of program's memory allotment). The *SPIWrite* function doesn't have a failing point (it is always executed successfully) so the result of the programming operation is calculated by reading the memory position and verifying the validity of the programmed words.

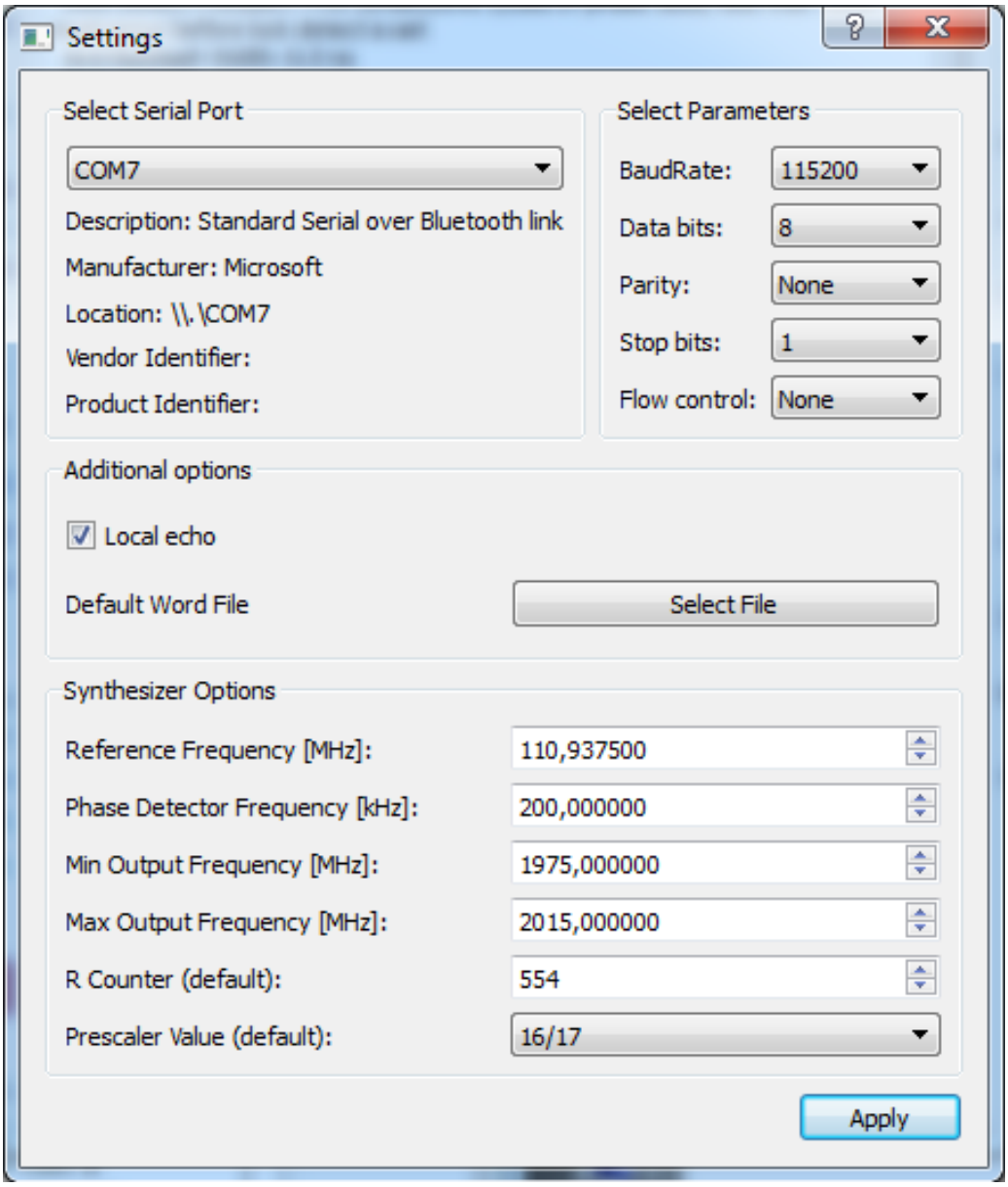


Figure 5.3: PC client window - settings

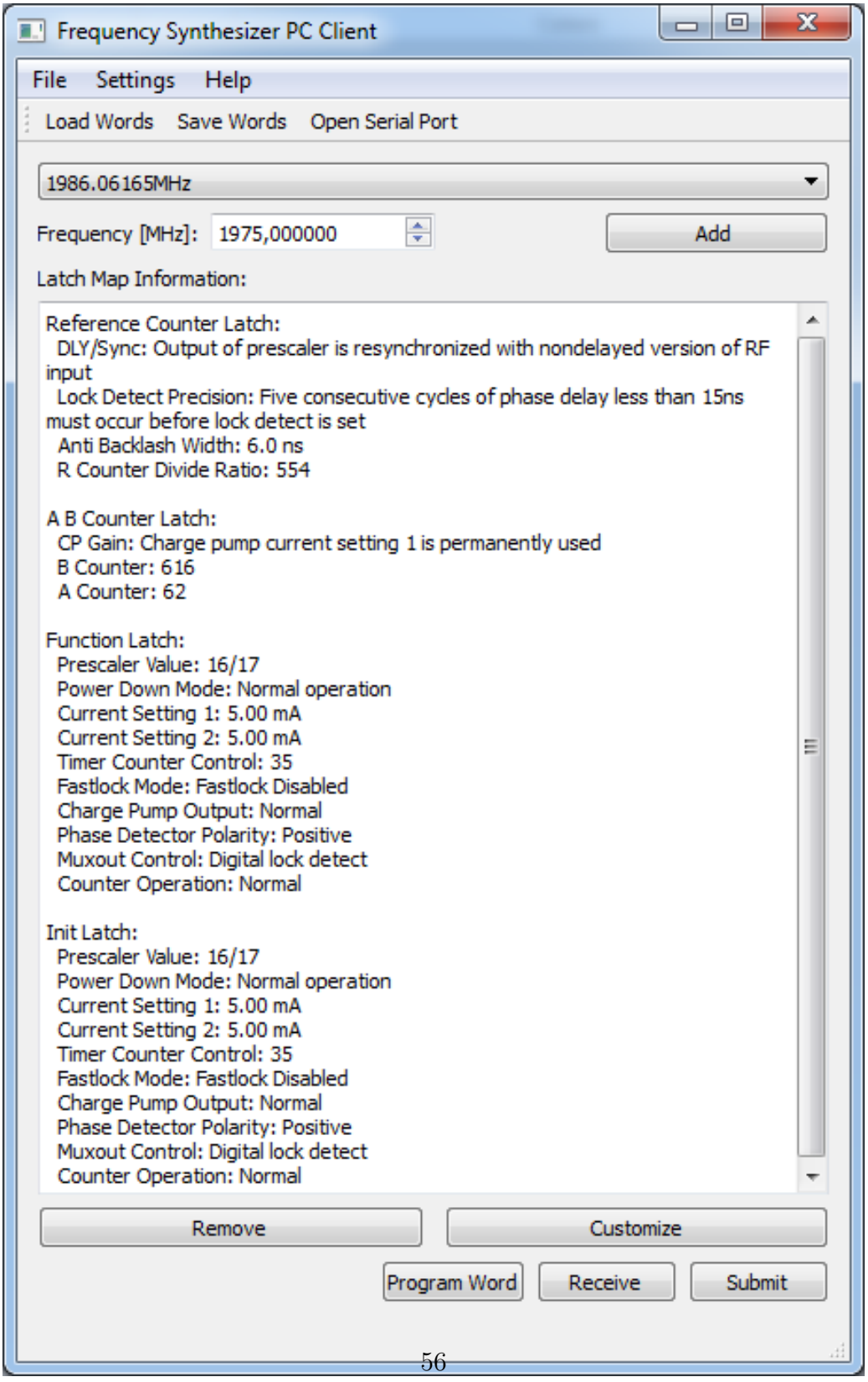


Figure 5.4: PC client window - main window

Chapter 6

PCB Design

As discussed in chapter 4, the hardware in this project consists of a board that can be reused with various PLL ICs (4.2.1) and different VCOs (4.2.3). Although, in order to demonstrate a working prototype, a specific board was built with a chosen PLL IC and VCO.

The design of the board was made using OrCAD software (OrCAD Capture for the initial schematic and OrCAD Layout for the PCB Design).

This chapter will be divided in three parts: the DC Circuit section (Section 6.1) details the power distribution part of the schematics. The Digital circuit section (6.1) explains the micro-controller portion of the overall circuit, and the High Frequency section (6.3), will detail the schematic of the PLL and VCO part of the circuit.

Figure 6.1 shows the final assembled PCB with all of the important components outlined. Table 6.1 enumerates the components outlined in the figure. These components will be further detailed below.

6.1 DC Circuit

The circuit is powered via a single 12 V line from a regulated power supply as adding a power supply to the PCB would add unnecessary complexity and interferences. The power is then fed to linear regulators that convert it to more usable voltages. In total, there are four linear regulators: two 5 V regulators and two 3.3 V.

The choice of using a seemingly high amount of regulators stems from the fact that the

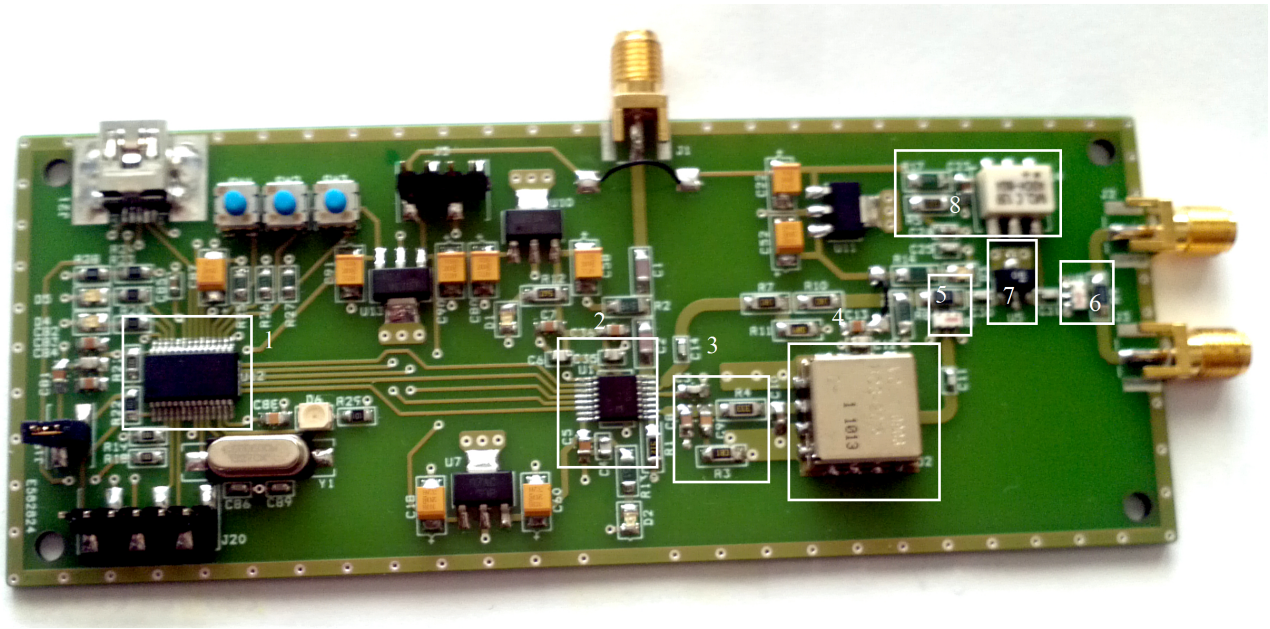


Figure 6.1: Final assembled device

PLL IC and VCO are very sensitive devices. An option was made to keep them as isolated as possible, and the fact that separate regulators are used helps in this regard.

One $LM29377$ 3.3 V regulator is used to power the micro-controller, while a second one powers the PLL's Digital section. For the PLL's analog power supply, and the VCO, two separate $LM2937$ 5 V linear regulators were used. This keeps both devices operating at very similar voltages but keeps them isolated from each other, which helps reduce interference.

6.2 Digital Circuit

The digital portion of the circuit is composed of the micro-controller and PLL IC. The micro-controller's part of the circuit is depicted in figure 6.2.

The micro-controller is programmed with the $J20$ connector using a PICkit 3 programmer. The MCU in use has USB support, however, the internal clock isn't always as precise as needed to operate the USB module (it differs from chip to chip). This is a know issue on these model of MCU. As such, a crystal was added to be used as a stable clock generator for the USB module. This was a compromise as adding more signal generators in the PCB would add to the

1. Micro-Controller;
2. Synthesizer;
3. Loop Filter;
4. Voltage Controlled Oscillator;
5. Power Splitter;
6. Power Splitter;
7. Amplifier;
8. RF Choke.

Table 6.1: Enumeration of figure 6.1

interferences, but one that was needed due to the requirements of the device. One argument to be made for the use of the crystal is that, because the micro-controller's relatively imprecise oscillator can now be turned off, we are left with only one source of interference which is more precise than without using the crystal.

Pins 10 to 13 of the micro-controller are used as the SPI interface for programming the PLL while pin 14 is used as the PLL's MUXOUT input. This pin is also used to power a LED that lets the operator know there was a phase lock and the synthesizer is now working at the expected frequency.

Pins 16 to 18 connect to the switches which can be used to operate the device, program and cycle through different frequency words. And pins 25 and 26 serve as the USB's status LEDs, that let the operator know if there was a USB error or if everything is working correctly.

6.3 High Frequency Circuit

The high frequency part of the circuit is depicted in figure 6.3. The PLL IC is $U1$ on the left, depicted as $ADF4112$, while the VCO is the $U2$ chip, ROS-2015+, the same as in the final device.

Pin 2 of the PLL component is the Charge Pump Output. It is connected to the VCO's

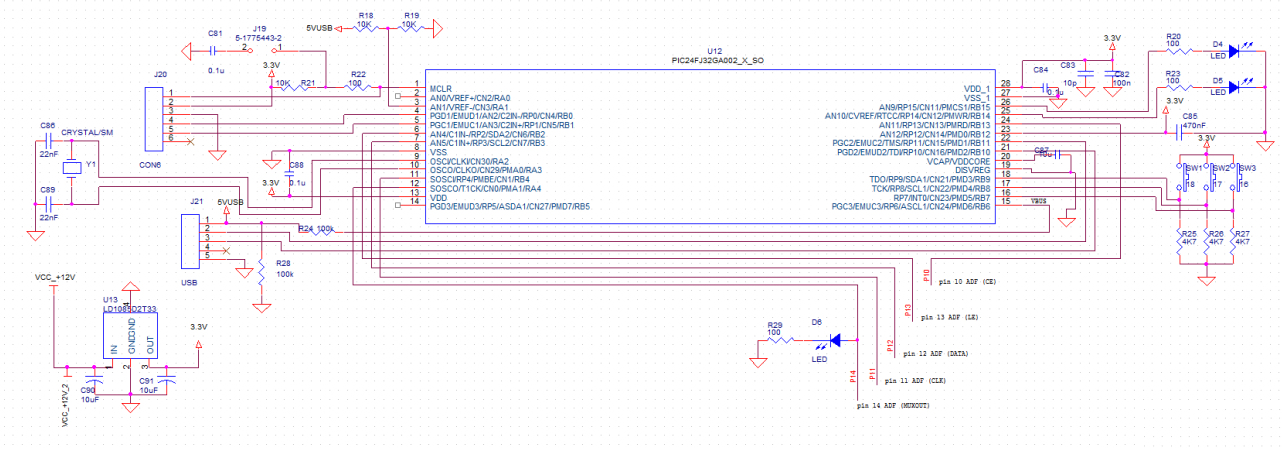


Figure 6.2: Spice diagram of the micro-controller section of the circuit

---

"V-Tune" pin through the Loop Filter (Components C8, C9, R3, R4, and C10). As we will see in section 6.3.2, these components were calculated with the help of Analog-Device's ADISim PLL Software. Appendix C contains the loop filter calculations for different output frequency ranges, also calculated by ADISim PLL.

The loop filter integrates the current pulses at the output of the synthesizer's charge pump and presents an approximately linear voltage at the input of the VCO's v-tune pin.

The chosen VCO has a typical output power of 7.5 dBm. This signal is then the input of a power splitter (SCN-2-22) which divides the signal into two separate signals, one for amplification and ultimately to the output, and another is fed back to the PLL IC to be used as input to the phase detector. A power figure of 7.5 dBm relates to an output power of $7.5[dBm] = 10\log_{10}\frac{P_{out}[mW]}{1[mW]}$, which gives us $P_{out} = 10^{\frac{7.5}{10}}1 = 5.6mW$.

Generally speaking, a power splitters' function is to divide the input signal to multiple outputs. This means that the signal power will be divided in two, if the signal is split into two separate signals. This means a power reduction of 3 dB at each ouput. Since no device is lossless, we also have to contemplate insertion loss of the device. According to the power splitter's data-sheet, it has a typical total loss of 3.4 dB at a frequency of 1980 MHz, which is the close to the frequency range used in the project.

The reduction of 3.4 dB at the output of the splitter means there is a signal power at each output of approximately 2.56 mW, or 4.08 dBm. This signal is then passed through an amplifier. The chosen amplifier was Mini Circuits Gali-6+. This amplifier has an operating range from

DC to 4 GHz, which means it is suitable for every possible frequency range of the final device.

According to the Gali-6+'s data-sheet, the amplifier present a gain of 11.80 dB at the frequency of 2 GHz. This means that we will theoretically have an output power of 38.75 mW or 15.89 dBm. This amplifier is biased with the use of an RF choke (ADCH-80A+). The function of the choke is to isolate the high frequency at the output of the amplifier (which is also the DC input) from the voltage source.

The biasing of the amplifier, according to the data-sheet, is done by imputing a 70 mA current at it's DC input line. Since the amplifier operates at a typical voltage of 5 V, this is the DC voltage we expect at it's output. Therefore we place a 100 Ω resistor between the device's VDD pin (12 V) and the choke input (which, as the amplifier, will present 5 V). This voltage difference of 7 V, with the 100 Ω will produce the necessary current of 70 mA. However, since a resistance of 100 Ω will cause a power loss of approximately $P = \frac{V^2}{R} = \frac{7^2}{100} = 0.49$. This means that this resistor will dissipate approximately 0.5 W. Because this power dissipation is close to the limit that the used resistors can handle, the biasing resistor was divided into two 200 Ω resistors placed in parallel.

The output of the amplifier is then once again divided so there are two possible output signals. As before, this will entail a reduction of 3.4 dB in each output, which in turn means that the final output signal will present a power of approximately 17.71 mW or 12.48 dBm.

6.3.1 Phase Frequency Detector

As previously discussed, the phase frequency detector compares two input signals. If these signals differ, the output will be proportional to the difference between the two input frequencies.

In the case of the chosen ADF synthesizers, the input reference signal passes by a programmable divider, referred to as "R Counter", which will divide the input reference by R. This signal is then fed into the phase detector. The other PFD input is the VCO output after being passed through a frequency divider referred to as the "N Counter". The N Counter is also programmable, but is configured by a set of different counters. The Pre-scaler, A and B Counters in the following relation: $N = BP + A$. As may be seen in figure 6.4.

The phase detector frequency is set by the value of the R counter. We get the PFD frequency by dividing the value of the input reference frequency by the value programmed in the R counter.

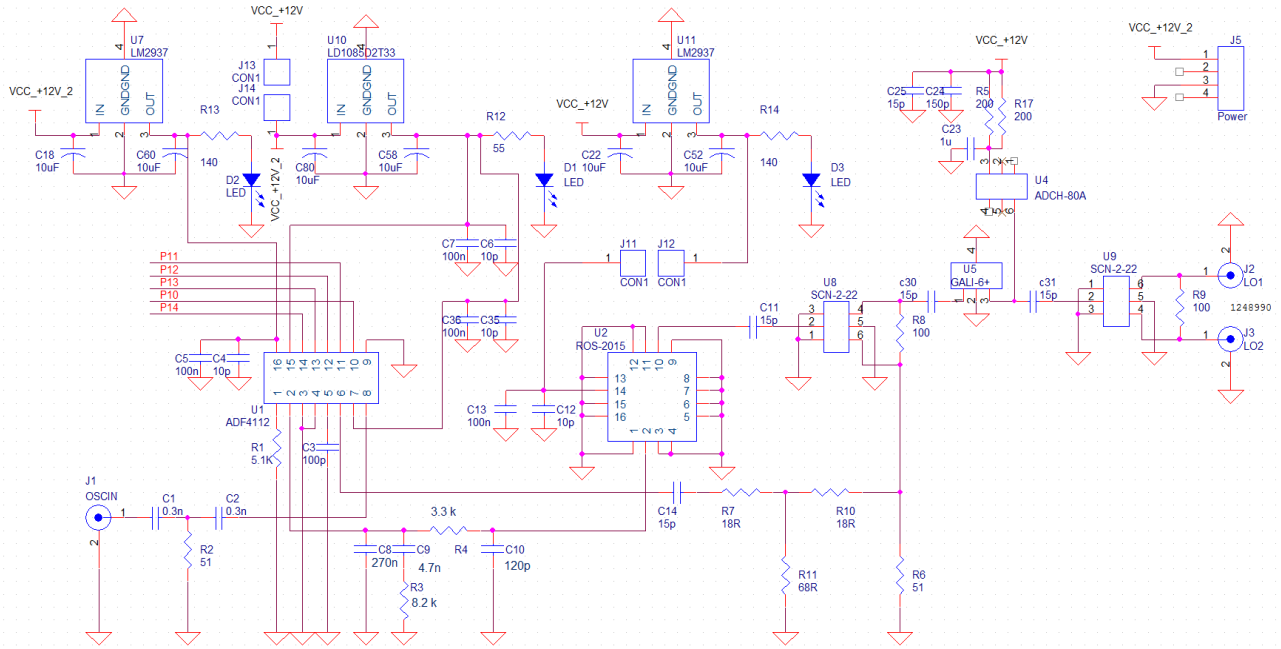


Figure 6.3: Spice diagram of the high frequency section of the circuit

---

$$f_{PFD} = \frac{f_{REF}}{R} \tag{6.1}$$

The output frequency of the phase detector is then fed into a loop filter in order to integrate the signal output to, hopefully, a stable voltage which, at the input of the VCO will tune it to the desired frequency. The final frequency will then be given by:

$$f_{out} = N \times f_{PFD} = \frac{[(P \times B) + A] f_{REF}}{R} \tag{6.2}$$

This equation may be analysed to say that the minimum frequency increment that may occur is the same as f_{PFD}, that is if N increases by one (N is always an integer number), the output frequency will be increased by the same as the PFD frequency. This means that the phase detector frequency dictates our channel spacing.

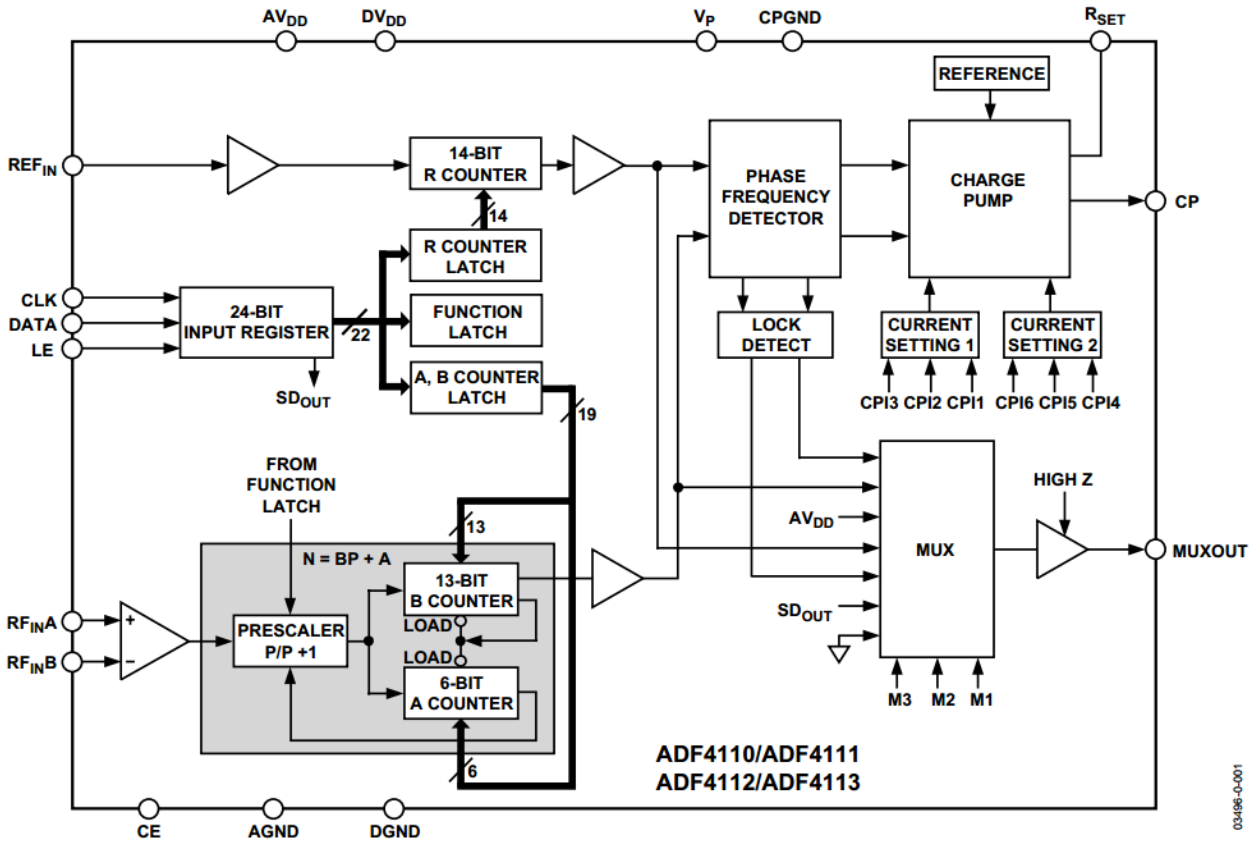


Figure 6.4: Functional block diagram of the ADF4110 family synthesizers from: [22]

6.3.2 Loop Filter

The loop filter is a crucial part of designing a Phase-Locked Loop synthesizer. The nominal values of the loop filter components are usually calculated by compromising between conflicting requirements.

A loop filter is there in order to remove unwanted frequency components between the phase detector output and the VCO tune line, as they would cause spurious frequencies in the VCO output.

However, a loop filter will also affect related characteristics such as the speed with which the output frequency may be changed, for instance a narrow band filter may cause for a very slow change in the VCO Tune line, which will in turn cause a slow change in the VCO's output. A filter with a wider band may allow for faster switching but may also cause unwanted spurious

frequency content in the output.

The most common type of loop filter is the third order integrator. According to [23], "In general, the loop filter bandwidth should be 1/10 of the PFD frequency (channel spacing)".

6.3.3 Calculation

As a compromise, the frequency chosen for the phase detector was 200 kHz. This is a frequency that allows for a relatively low channel spacing (giving us approximately possible frequency channels within the 40 MHz of output bandwidth) as well as a low noise level on the output.

Analog Devices, the manufacturer of the chosen synthesizer chips, provides a very useful tool for calculating loop filters. The ADISimPLL v3.60 software was used to make the calculations in this project.

The parameters chosen were:

- A minimum output frequency of 1975 MHz;
- A maximum output frequency of 2015 MHz;
- A channel spacing (PFD frequency) of 200 kHz;
- The chosen VCO was the ROS-2015+;
- A reference frequency of 104 MHz;
- A loop bandwidth of 10 kHz.

The results, after adapting the capacitor and resistor values for more common ones may be seen in figure 6.5. The change of the theoretical values for real-life, common values changes the theoretical loop bandwidth to 10.2 kHz, which is similar to the one used in the design.

Figure 6.6 shows the phase noise prediction calculated by ADISimPLL.

Annex C details further calculations needed to produce devices with different output frequencies, while utilizing the board produced for this project.

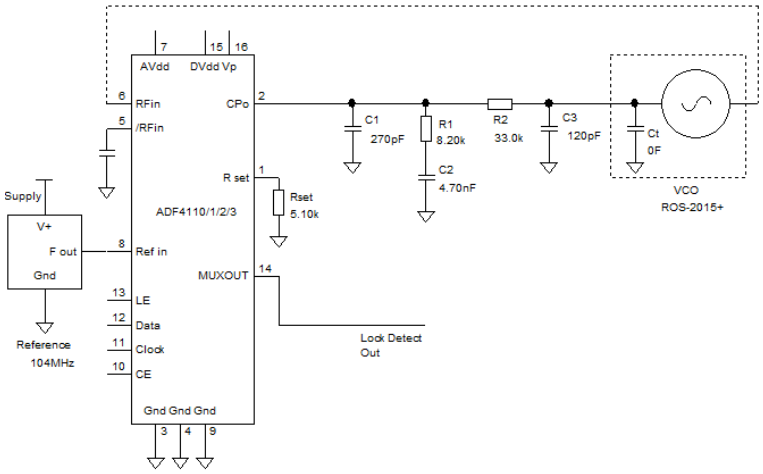


Figure 6.5: Loop filter schematic as calculated by ADSimPLL

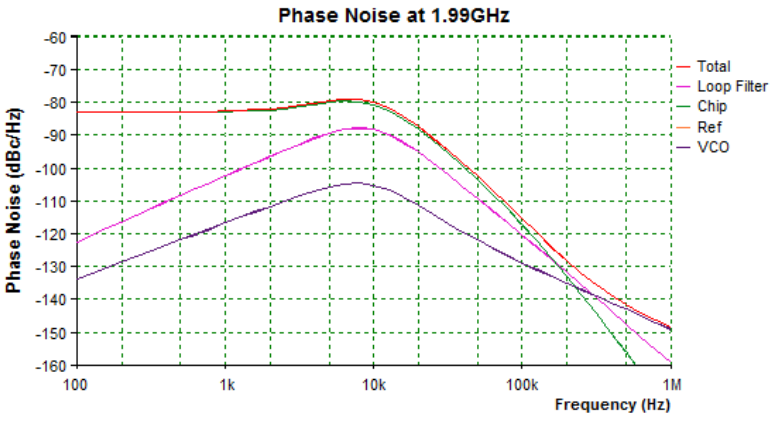


Figure 6.6: Phase noise as calculated by ADSimPLL

Chapter 7

Results

7.1 Signal Quality

7.1.1 Expected Results

As it relates to signal performance, the objectives set for this project are simply to present signal quality comparable to other devices of the same kind. The term signal quality is used here to refer to the output power of the main spurious content of the output signal, and phase noise performance.

Looking at the data-sheet for the ADF4110 family, the synthesizer chip utilized presents the phase noise characteristics shown in 7.1. These two figures, show us the amount of output power there is at certain frequency distance points from the carrier.

The spurious content of the same device is shown in 7.2.

The phase noise at a distance of 200 kHz of the carrier, shown in figure 7.3, is of approximately $-90.2dB$ relative to carrier.

These measurements are a good starting point for an estimation of the final results of the device.

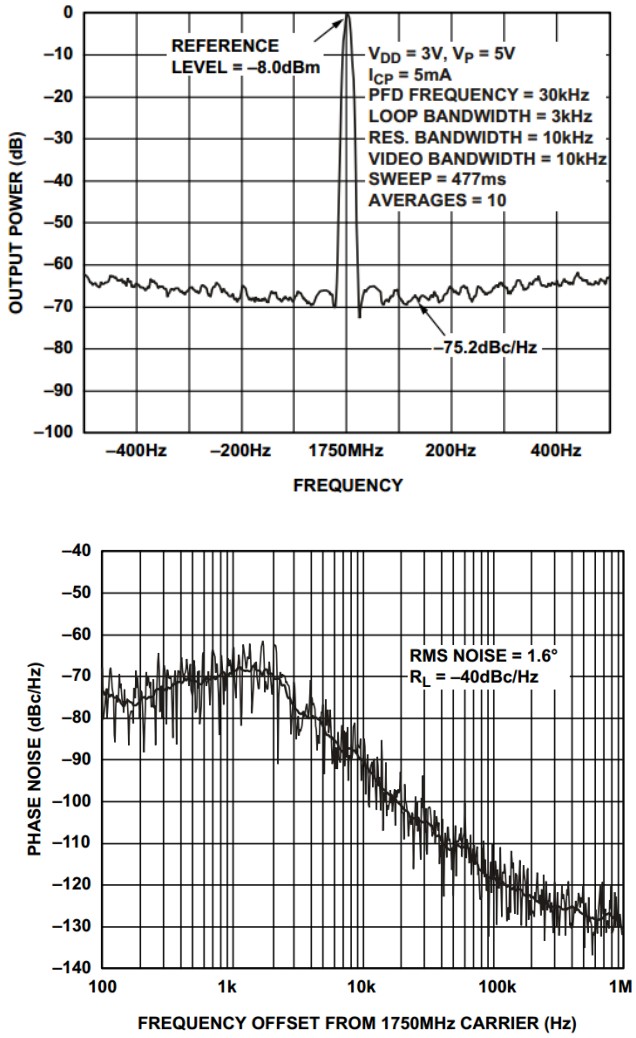


Figure 7.1: ADF4113 Phase Noise [22]

7.1.2 Actual Results

To make the measurements, the finished device was connected to the USB port of the PC and the client was opened. The frequency of 1981.05 MHz was chosen as a middle of band frequency to perform the measurements. Figure 7.4 shows the output signal's frequency content after programming the device to 1981.05 MHz. As we can see, the closest frequency we can achieve to this frequency was programmed to the PLL by the micro-controller and the PLL was correctly able to hold the frequency in lock at approximately 1981.049 MHz.

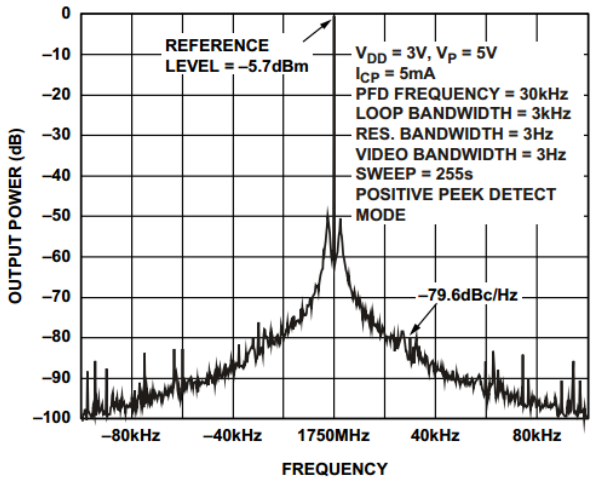


Figure 7.2: ADF4113 Reference Spurs [22]

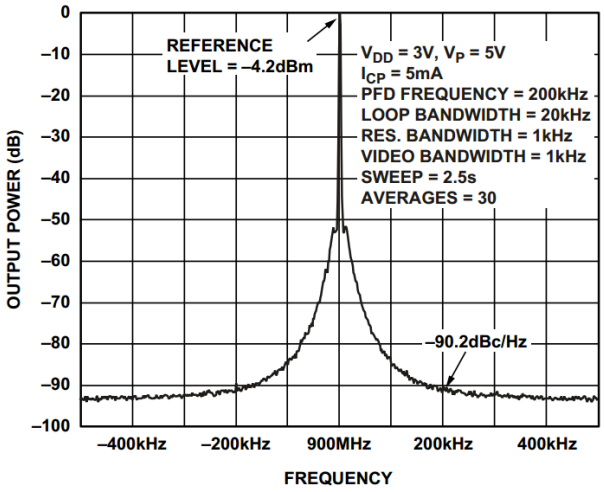


Figure 7.3: ADF4113 Noise Floor [22]

Output Power The theoretical output power calculated in 6.3 points to a value of 12.48 dBm or 17.71 mW. In reality the first device produced had a soldering problem in the power divider, which caused a very low power in the output value.

If we consider the same circuit without the amplifier we have the VCO output at the same 7.50 dBm (5.60 mW), but this signal is now reduced twice by the power splitters, which give us a reduction of 3.40 dB as stated before. This results in an output power of 1.17 mW or 0.68 dBm.

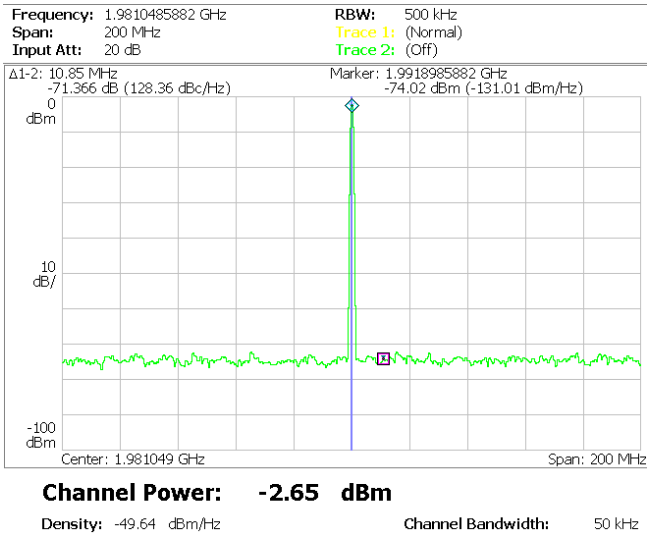


Figure 7.4: Output signal at 1981.05 MHz, span 200 MHz

As can be seen in figure 7.5, the actual output power of the device is closer to $-2.71\,\text{dBm}$, which relates to $0.54\,\text{mW}$. This difference may be explained by the insertion losses of the components which were not considered in all cases in the calculations.

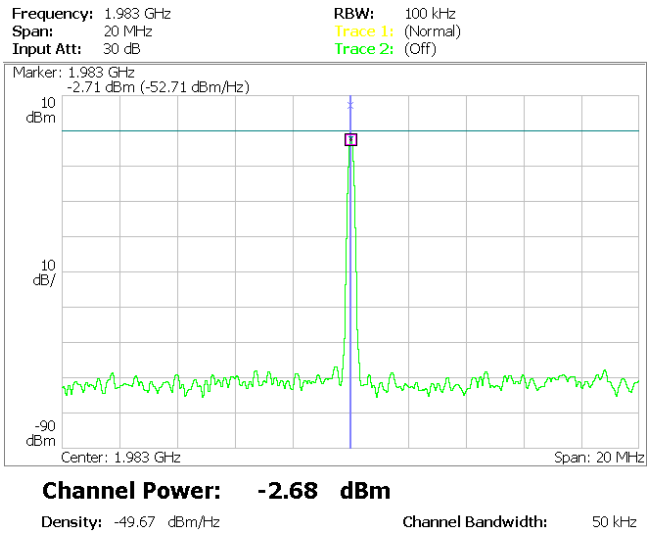


Figure 7.5: Output signal at 1983.05 MHz, span 20 MHz, first device

In the second device, the signal amplification problem did not occur, as we can see in figure 7.6. This shows an output power of approximately $10\,\text{dBm}$. This is an improvement of

approximately 12.5 dBm over the first board with the defective amplifier.

This value is still short of the theoretical value, but the difference may once again be explained by insertion losses of the components not considered in the theoretical analysis, as well as the real-life differences between typical, or theoretical, and real loss figures.

The second device produced, however, had a problem in the loop filter, where incorrect values were soldered. This problem was not solved in time for the new measurements to be included in this thesis.

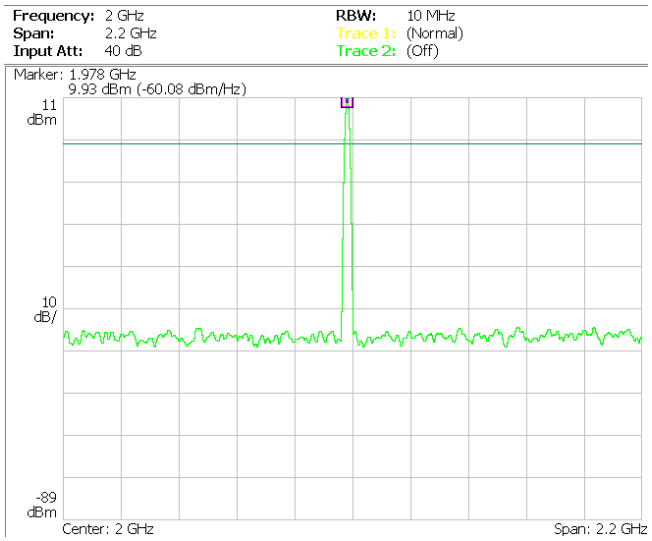


Figure 7.6: Output signal at 1978 MHz, second device

Phase Noise Performance As shown in figure 7.4, we get a signal with a noise floor of approximately $-71.4dBm$. In figure 7.7, we can see that at a distance of approximately 5.47 Hz we have a power of 46.36 dBm and a phase noise value of -43.10 dBc.

The final performance of the device, as it pertains to the signal characteristics, was not close to the expected from the information included in the synthesizer chip's data-sheet. This is may be due to the lack of a functioning amplifier but is certainly also due to differences in measuring equipment. It is very difficult to accurately measure phase noise due to the fact that any measuring tool will introduce its own noise in the measuring operation.

Figure 7.8 shows the output signal at a span of 1MHz, with a configured frequency of approximately 2003.8 MHz. We can see in figure 7.9 that the power at a distance of 200 kHz

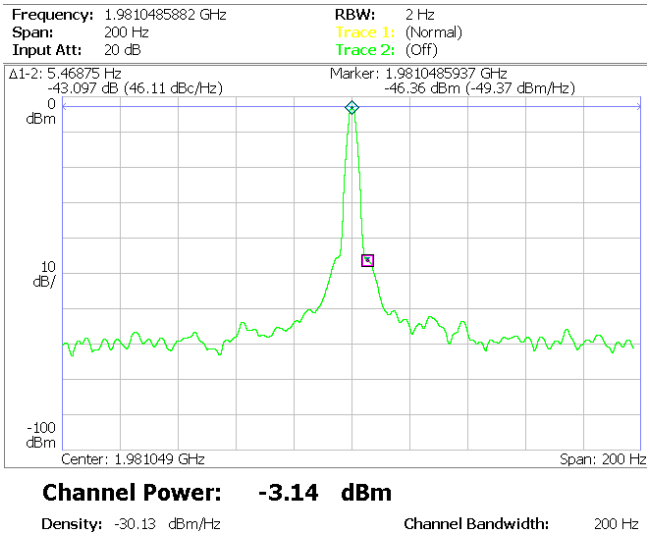


Figure 7.7: Output signal at 1981.05 MHz, span 200 Hz

---

is of approximately $-74.4\,\mathrm{dBm}$.

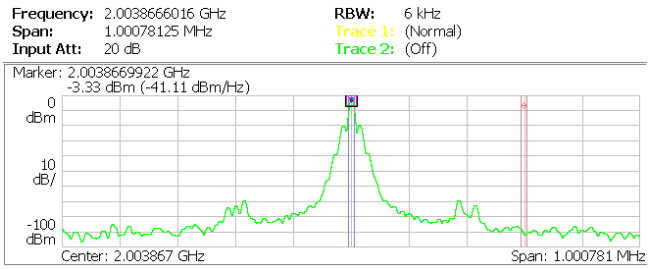


Figure 7.8: Output signal at 2003.8 MHz, span 1 MHz

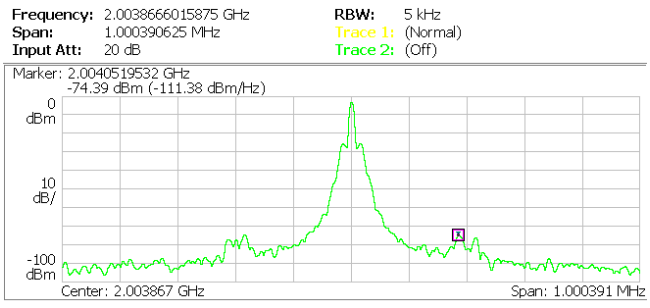


Figure 7.9: Output signal at 2003.8 MHz, span 1 MHz

Spurious Figure 7.10 shows the harmonic signals generated by the device. These frequency spikes are experienced at approximately ±184.2 kHz and ±215.4 kHz.

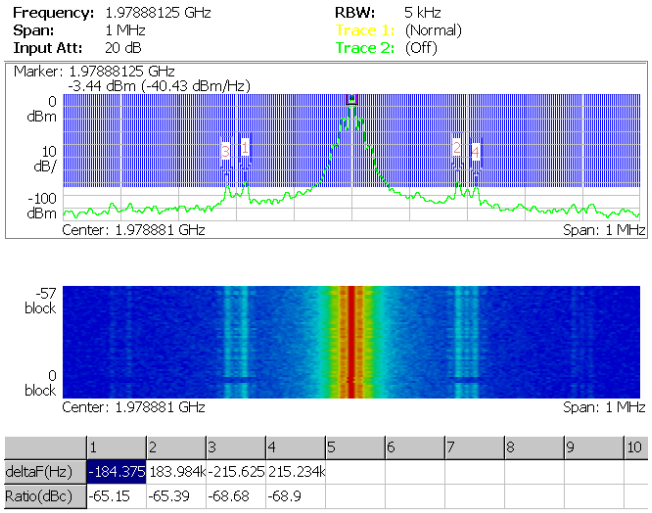


Figure 7.10: Spurious - output signal at 1978.88 MHz, span 1 MHz

Harmonics The harmonic content of the device may be seen in figure 7.11. The configured output frequency in this case is 1979 MHz and the output power is −2.94 dBm which translates to 0.51 mW.

As we can see in the figure, there is an harmonic at twice the output frequency (3958 MHz) with a decrease in output power relatively to the main frequency component of approximately 20 dBm. This equates to a reduction in output power of −19.57 dB. The data-sheet of the oscillator used in the project ([24]) presents typical harmonic values in dBc (dB relative to carrier) of −27.70 dB for an output frequency of 1981.6 MHz (the closest documented value).

The second harmonic occurs at 5937 MHz, three times the main output signal, with a power reduction of approximately 30 dBm, which translates to a power loss of −29.61 dB, where the data-sheet predicts a reduction of −33.40 dB

The final harmonic frequency measurable with the equipment available occurs at four times the main output frequency (7916 MHz), with a power reduction of approximately 40 dBm. Which relates to a reduction of −38.48 dB. The data-sheet predicts a power loss at this harmonic of −42.50 dB

Figure 7.11: Harmonic content of the output signal

Chapter 8

Conclusions and future work

8.1 Lessons Learned

8.1.1 PC Client

In order to ease the cross-platform development of the PC Client, and because initial development was done using Linux as an operating system, the QT Framework was chosen for development. While this framework is very powerful and cross-platform, programs written in C++ are compiled into machine code, which means they need to be compiled for every architecture the program is to be used on. This makes it so the finished program will need extra files (such as DLLs on Windows), and because the framework sits on top of the standard C++ framework, it is a complex process to implement custom components.

However, QT offers a wide array of widgets that may be configured to do what the developer needs. It has a comprehensive set of documentation available both on the Software Development Kit (SDK) and the Internet in general. Although it is sometimes difficult to change from the different compilers used in Linux (g++) and Windows (MSVC), the QT framework does a good job of easing the transition by wrapping native widgets and code in a common API across platforms.

There are alternatives, such as using Java's Swing framework, or Microsoft's own .NET platform. The advantages of using Swing to develop programs is that they are cross-platform as long as there is a Java installation present in the target system. There is no need to ship extra

libraries as the Java runtime environment contains everything that may be necessary. However, the general consensus around Swing is that it is harder to understand and use for someone with little to no experience in Java and although there are large repositories of documentation for Swing, its development has been slow in recent years.

Microsoft's .NET framework is quite flexible and well supported in Windows environments. Doing initial research for the project, it was found that because of this framework's poor to non-existent Linux support (at the time of writing), it would entail a complete re-write of the program if it ever was to work on Linux or indeed on Apple's Mac OS. The .NET framework supports a multitude of programming languages, however, development in the C# language is the norm for graphical programs. As this project was meant as a learning exercise as well as all of the previously stated goals, and because the C++ language entails diving deeper into concepts like pointers, memory management and other low level abilities which are abstracted in the .NET's implementation of C#, it was deemed that doing the program in the C++ language would bring the highest challenges, but also the biggest rewards.

The final decision was then between the .NET and the QT frameworks. Both are good frameworks, but QT was chosen in the end because it filled the requisite of being easy to develop cross platform and using a language that is important to learn for future usage. The difficulties of learning C++ and QT became apparent early in the development stage. As expected, the learning process was highly challenging but also yielded good rewards, as the developer's programming skills came out greatly improved in the end. It would probably have been easier and less time consuming to develop the application using .NET but the decision to use QT turned out the be the correct one.

8.1.2 Microchip's PIC

Because of previous experience using Microchip's PIC micro-controllers, these were chosen to develop the hardware. Although the PIC micro-controllers are good, inexpensive devices that are perfectly capable of doing the job they are intended to do, Microchip's documentation seems, in the developer's opinion, to be overly complex.

Although there are good code examples in Microchip's website that can be adapted to work in its devices, this process is not simple. Microchip's provided data-sheets are often outdated and in some cases contain incorrect information that may lead developers to error.

As projects like the Arduino begin to take traction, it might have been a wiser choice to use an Atmel micro-controller as there is extensive on-line support and documentation available, especially as it pertains to the USB software stack and capabilities of the hardware, which was undoubtedly the main point of difficulty during the development of the project.

There is a stability problem with the two boards produced in that random power fluctuations cause the micro-controller to momentarily stop working. There was not enough time to properly investigate the origin of the issue while writing this report. However, for most of the duration of the project, the developer believed that this problem stemmed from the USB stack being used. The reasoning behind this is that these power fluctuations are only apparent when the device is connected to the USB port in the computer as the connection is momentarily lost. This, allied with the fact that the basic CDC drivers in Windows do not recover well from these type of problem, means that the USB connection must be re-established when these fluctuations occur.

8.2 Future Work

This thesis was written around the development of an easily changeable platform, both in the hardware and software portions of the work, that is capable of easily being modified to allow for different uses and frequency outputs. While there were two devices built, they both used the same components and frequencies, in order to better analyse the output signal's parameters like phase noise.

As explained before, there was a power fluctuation problem in the devices assembled that momentarily interrupts the PC communication. This makes it so the user has to unplug and plug back in the USB cable to the device. The first step to solve this problem would be to investigate the underlying cause. This, sadly, was not done in the scope of this project due to time constraints.

To better demonstrate the value of the project, the next step would be to develop hardware for different frequency intervals. As such, annex C was included with the calculations necessary to produce new versions of the device for different output frequencies.

A few additions would allow this project to be more useful for all intended applications. First, since the micro-controller used has support for USB OTG, it is relatively straightforward to add support for other USB devices to be connected to it. Allowing that the device store log

files in a USB pen drive for instance. It is also not complex to add support for LCD screens so the user can better control the device and know exactly its status at all times.

Another addition may be a frequency scanner module, where the device increments through stored programming maps and programs them at given intervals to the PLL. This may be useful in searching for modulating frequencies of signals while utilizing external hardware.

If a PCB re-design was to be made, the device would have increased signal performance if more shielding was included between and around the high frequency lines, so as to reduce susceptibility to noise. It would also benefit from shorter paths in the signal lines.

8.3 Conclusion

This project mainly focused on the development of an hardware platform for controlling a frequency synthesizer using only a common computer as a tool, and all of the surrounding software needs.

As detailed in chapter 1, the main objectives set for this paper are the following:

1. That it is less expensive to produce than the available off-the-shelf synthesizers in existence;
2. That its output signal quality is suited for use in laboratory, field and hobbyist settings;
3. That it is easy to use. This includes being easy and fast to program and deploy without using external hardware or software;
4. That it can easily be tailored to work with other frequencies by changing only a few components;
5. That it provides an output power of about 7 dBm to be used as a local oscillator for double balanced ring diode mixers.

Cost: For this project, only two boards were produced, but to accurately compare the manufacturing cost with the off-the-shelf devices analysed in chapter 1, we will calculate the cost necessary for the production of 20 boards, keeping in mind that for any production run over this value would only bring the cost of manufacture down.

By far the most costly component included in the device is the VCO. The chosen Mini-Circuits ROS-2015+ has a unit cost, through the manufacturer's website, of 18.46€ with a purchase of 20 or more units [25].

The second most expensive device is the micro-controller, which is sold for 6.89€ for 20 units [26].

The ADF4112 synthesizer has a unit cost of 2.68€ with a purchase of 96 or more units [27].

The RF choke (ADCH-80A+) has a unit price of 2.55€ with a purchase of 20 or more units [28].

The Gali-6+ amplifier is sold with the price of 1.38€ with a purchase of 20 or more units [29].

Allowing 5.00€ for the remaining - less expensive - components (e.g. resistors, leds and capacitors), we have a final component price per device of approximately 40€.

The PCB in itself has a cost of approximately 8.10€ per unit for a purchase of 20 units according to [30]. Thus bringing our total price per device to under 50€.

Since the price of the PCB raises significantly if there are less units printed (around 60€ for a single unit) the cost for building and assembling a single unit would come to approximately 110€.

The figure of 110€ for a single device is less then half the price of the least expensive alternative found currently on the market, and although the device may not have all the features of the more expensive alternatives, it presents a high reduction in cost for a user willing to build his own device, for his own needs.

Ease of Use: The project focused a high amount of effort on making the PC client software as straight-forward to use as possible while still allowing some freedom for more experienced users.

The plug-and-play nature of the USB connection used allows for portability and simplifies the use and programming of the device for any user with a PC, without the need for external hardware or software.

Descending to the implementation level, the communication protocol was made to be easy to use and customize, giving the user the ability to use external software if he so desires to implement a more complicated use for the device. For instance a user may create a script or

program which will periodically change the output frequency of the device, or implement a sweep program which will change the device's output frequencies inside a range with specific frequency steps and at specific time intervals.

Customizability: The device was assembled using a specific synthesizer and voltage controlled oscillator, but it was created to allow for other components of the same families. As explained in 4.2.3, it is possible to use many other VCOs from the large array of devices belonging to the Mini-Circuits ROS family. The ADF4110 family of synthesizers also covers a large frequency range as shown in 4.2.1. Appendix C includes some examples and instructions on how to calculate loop filter components necessary for a change in output frequency range.

The power splitter used in the project (SCN-2-22) has a somewhat limited working frequency range, from approximately 1850 MHz and 2200 MHz. This makes it so that this component also has to be replaced if the intended output frequency range exceeds its limitations. This device can be replaced by another splitter of the same family, or simply by a resistive power splitter, which could, however, entail a PCB re-design. The SCN family of devices from mini-circuits has footprint compatible splitters from 800 MHz to 2700 MHz.

All these considerations make the device easy to customize, in order to serve as a solution to many of the applications that may be needed for use in laboratory, field or hobbyist settings.

Bibliography

[1] Quonset Microwave. *USB Stick Synthesizer*. [Online; accessed 28-September-2014]. 2014. URL: http://www.quonsetmicrowave.com/category-s/54.htm/.

[2] LLC. Windfreak Technologies. *Low Cost RF Building Blocks*. [Online; accessed 28-September-2014]. 2014. URL: http://www.windfreaktech.com/synthusb.html.

[3] sdr kits. *SDR-Kits*. [Online; accessed 28-September-2014]. URL: http://sdr-kits.net/QRP2000_Description.html.

[4] Fredrick Matos. *Radio Frequency Spectrum Management and Time and Frequency Standards*. Ed. by Wendy M. MiddletonMac E. Valkenburg. Ninth Edition. Woburn: Newnes, 2002. ISBN: 978-0-7506-7291-7. DOI: http://dx.doi.org/10.1016/B978-075067291-7/50003-0. URL: http://www.sciencedirect.com/science/article/pii/B978075067291750003

[5] *Mini-Circuits VCO Designer's Handbook*. 1996.

[6] Gargan Wikimedia Commons. *Common base Colpitts oscillator*. [Public Domain; Online; accessed 28-September-2014]. URL: http://commons.wikimedia.org/wiki/File:Cb_colp.svg.

[7] Omegatron Creative Commons. *Clapp oscillator*. [Creative Commons; Online; accessed 28-September-2014]. URL: http://commons.wikimedia.org/wiki/File:Clapp_oscillator.png.

[8] Gargan Wikimedia Commons. *Common base Colpitts oscillator*. [Public Domain; Online; accessed 28-September-2014]. URL: http://commons.wikimedia.org/wiki/File:Cc_colp2.svg.

[9] *ROS-70-219+ Datasheet*. [Online; accessed 28-September-2014]. URL: www.minicircuits.com/pdfs/ROS-70-219+.pdf.

[10] Jacques Audet. «ARRL - QEX - Q Factor Measurements on L-C Circuits». In: (2012). [Online; accessed 28-September-2014]. URL: http://www.arrl.org/.

[11] www.rfic.co.uk. *RF, RFIC and Microwave Theory, Design - Phase Noise*. [Online; accessed 28-September-2014]. URL: www.rfic.co.uk.

[12] *Mini-Circuits booklet: VCO Phase Noise*. 1999.

[13] Iulian Rosu. *Booklet: Phase Noise in Oscillators*. URL: http://www.qsl.net/va3iul/Phase%20noise%20in%20Oscillators.pdf.

[14] Watkins-Johnson Company. *Tech-Notes: Local Oscillator Phase Noise (and its effect on Receiver Performance)*. 1981.

[15] J.R. Smith. *Modern Communication Circuits*. McGraw - Hill Higher Education, 1998.

[16] Texas Instruments. *Fractional / Integer-N PLL Basics*. Ed. by Curtis Barrett. Technical Brief. 1999, p. 7.

[17] *Analog Dialogue: Ask The Application Engineer — 33*. [Online; accessed 28-September-2014]. URL: http://www.analog.com/library/analogdialogue/archives/38-08/dds.html.

[18] Floyd M. Gardner. *PLL Frequency Synthesizers*. John Wiley and Sons, Inc., 2005. ISBN: 9780471732693. DOI: 10.1002/0471732699.ch15. URL: http://dx.doi.org/10.1002/0471732699.ch15.

[19] Brendan Daly. *Booklet: Comparing Integer-N And Fractional-N Synthesizers*. 2001.

[20] *Farnell*. [Online; accessed 28-September-2014]. URL: www.farnell.com.

[21] *Microchip Technologies*. [Online; accessed 28-September-2014]. URL: www.microchip.com.

[22] Analog Devices. *ADF4112 Datasheet*. [Online; accessed 28-September-2014]. URL: http://www.analog.com/static/imported-files/data_sheets/ADF4110_4111_4112_4113.pdf.

[23] *Analog Dialogue: Ask the Applications Engineer - 30*. [Online; accessed 28-September-2014]. URL: http://www.analog.com/library/analogDialogue/archives/36-03/pll/.

[24] *ROS-2015+ Datasheet*. [Online; accessed 28-September-2014]. URL: www.minicircuits.com/pdfs/ROS-2015+.pdf.

[25] Mini-Circuits. *Ordering, Pricing and Availability - ROS-2015+*. [Online; accessed 28-September-2014]. URL: http://217.34.103.131/MCLStore/ModelPriceDisplayhtml.

[26] Farnell. *MICROCHIP PIC24FJ32GB002 - Pricing*. [Online; accessed 28-September-2014]. 2014. URL: http://pt.farnell.com/microchip/pic24fj32gb002-i-so/mcu-16bit-pic24-32mhz-soic-28/dp/1846947.

[27] Analog Devices. *ADF4112 Datasheet and product info*. [Online; accessed 28-September-2014]. 2014. URL: http://www.analog.com/en/rfif-components/pll-synthesizersvcos/adf4112/products/product.html.

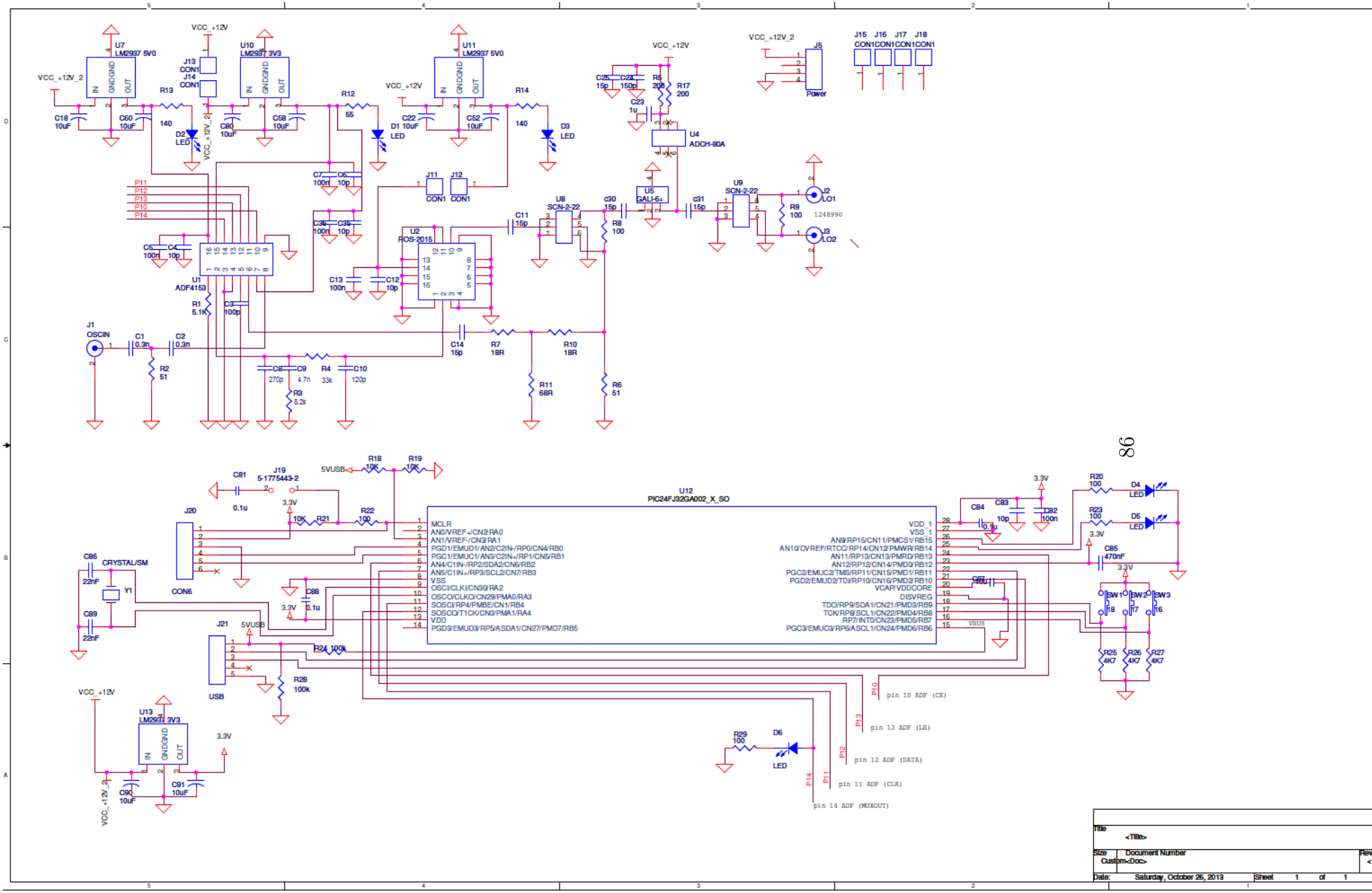
[28] Mini-Circuits. *Ordering, Pricing and Availability - ADCH-80A+*. [Online; accessed 28-September-2014]. 2014. URL: http://217.34.103.131/MCLStore/ModelPriceDisplay?14145346141240.20126717206542066.

[29] Mini-Circuits. *Ordering, Pricing and Availability - GALI-6+*. [Online; accessed 28-September-2014]. 2014. URL: http://217.34.103.131/MCLStore/ModelPriceDisplay?14145347883890.34172524700389806.

[30] Euro-Circuits. *PCB Price Calculator*. [Online; accessed 28-September-2014]. 2014. URL: http://be.eurocircuits.com/shop/orders/configurator.aspx.

Appendix A

Schematic and PCB


U7 LM2937 5V0
U10 LM2937 3V3
U11 LM2937 5V0
U13 LM2937 3V3
VCC_+12V
VCC_+12V_2
J13 CON1
J14 CON1
R13 140
R12 55
R14 140
D2 LED
D1 LED
D3 LED
C18 10uF
C60 10uF
C80 10uF
C58 10uF
C22 D1 10uF
C52 10uF
C25 15p
C24 150p
C23 1u
R5 205
R17 200
U4 ADCH-80A
J5 Power
J15 CON1
J16 CON1
J17 CON1
J18 CON1
U5 GALI-6+
U8 SCN-2-22
U9 SCN-2-22
C30 15p
C31 15p
C11 15p
R8 100
R9 100
1248990
J2 LO1
J3 LO2
J11 CON1
J12 CON1
C7 100n
C6 10p
C36 100n
C35 10p
C5 100n
C4 10p
U1 ADF4153
U2 ROS-2015
C13 100n
C12 10p
R1 5.1K
C3 100p
J1 OSCIN
C1 0.3n
C2 0.3n
R2 51
C8 270p
C9 4.7n
R4 33k
C10 120p
R3 8.2k
C14 15p
R7 18R
R10 18R
R11 68R
R6 51
P11
P12
P13
P10
P14
J19 5-1775443-2
C81 0.1u
5VUSB
R18 10K
R19 10K
J20 CON6
R21 10K
R22 100
C86 22nF
C89 22nF
CRYSTAL/SM
Y1
C88 0.1u
J21 USB
R24 100k
R28 100k
U12 PIC24FJ32GA002_X_SO
MCLR
AN0/VREF+/CN2/RA0
AN1/VREF-/CN3/RA1
PGD1/EMUD1/AN2/C2IN-/RP0/CN4/RB0
PGC1/EMUC1/AN3/C2IN+/RP1/CN5/RB1
AN4/C1IN-/RP2/SDA2/CN6/RB2
AN5/C1IN+/RP3/SCL2/CN7/RB3
VSS
OSCI/CLKI/CN30/RA2
OSCO/CLKO/CN29/PMA0/RA3
SOSCI/RP4/PMBE/CN1/RB4
SOSCO/T1CK/CN0/PMA1/RA4
VDD
PGD3/EMUD3/RP5/ASDA1/CN27/PMD7/RB5
VDD_1
VSS_1
AN9/RP15/CN11/PMCS1/RB15
AN10/CVREF/RTCC/RP14/CN12/PMWR/RB14
AN11/RP13/CN13/PMRD/RB13
AN12/RP12/CN14/PMD0/RB12
PGC2/EMUC2/TMS/RP11/CN15/PMD1/RB11
PGD2/EMUD2/TDI/RP10/CN16/PMD2/RB10
VCAP/VDDCORE
DISVREG
TDO/RP9/SDA1/CN21/PMD3/RB9
TCK/RP8/SCL1/CN22/PMD4/RB8
RP7/INT0/CN23/PMD5/RB7
PGC3/EMUC3/RP6/ASCL1/CN24/PMD6/RB6
VBUS
C84 0.1u
C83 10p
C82 100n
C87 0.1u
C85 470nF
R20 100
R23 100
D4 LED
D5 LED
SW1
SW2
SW3
R25 4K7
R26 4K7
R27 4K7
R29 100
D6 LED
pin 10 ADF (CE)
pin 13 ADF (LE)
pin 12 ADF (DATA)
pin 11 ADF (CLK)
pin 14 ADF (MUXOUT)
C90 10uF
C91 10uF
Title
<Title>
Size
Custom<Doc>
Document Number
Date: Saturday, October 26, 2013
Sheet 1 of 1

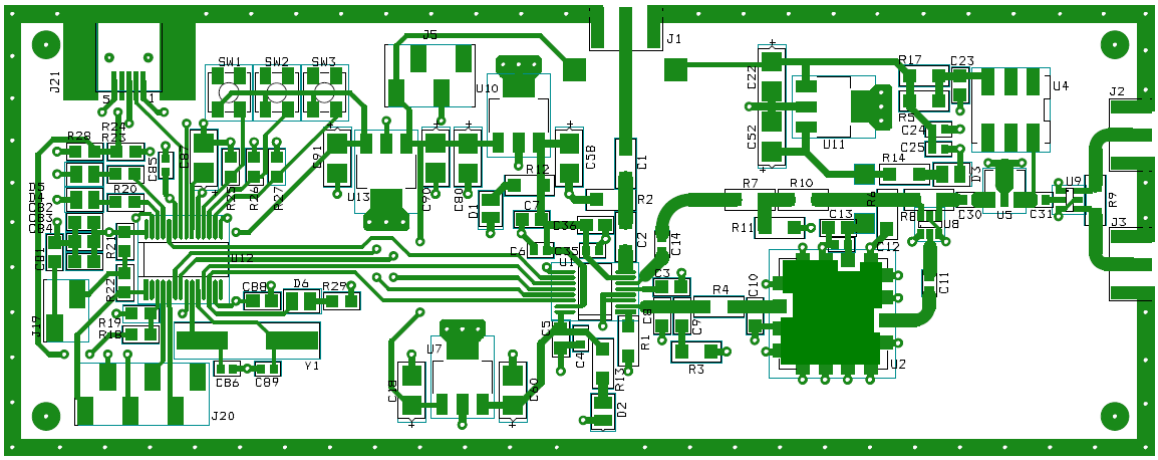


Figure A.2: Front layer of the PCB design

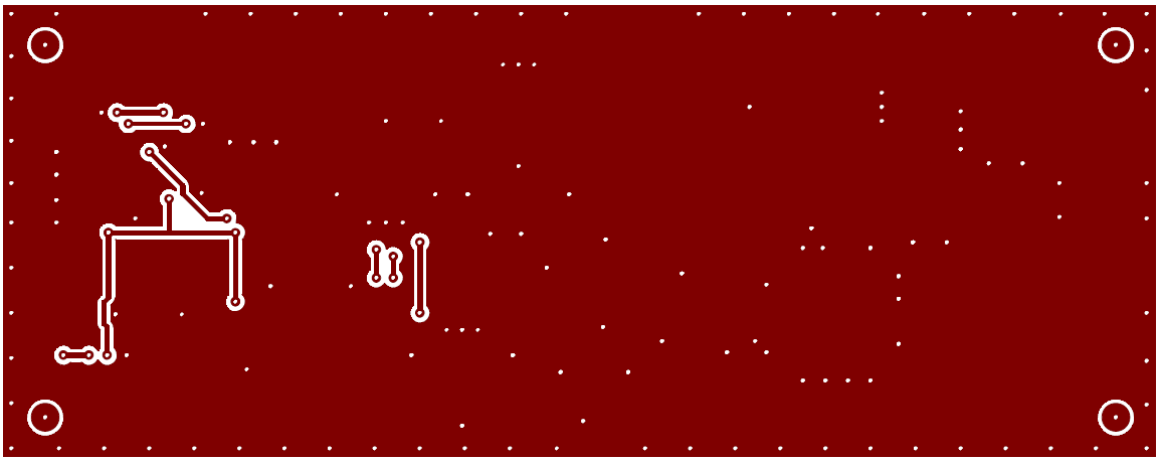


Figure A.3: Back layer of the PCB design

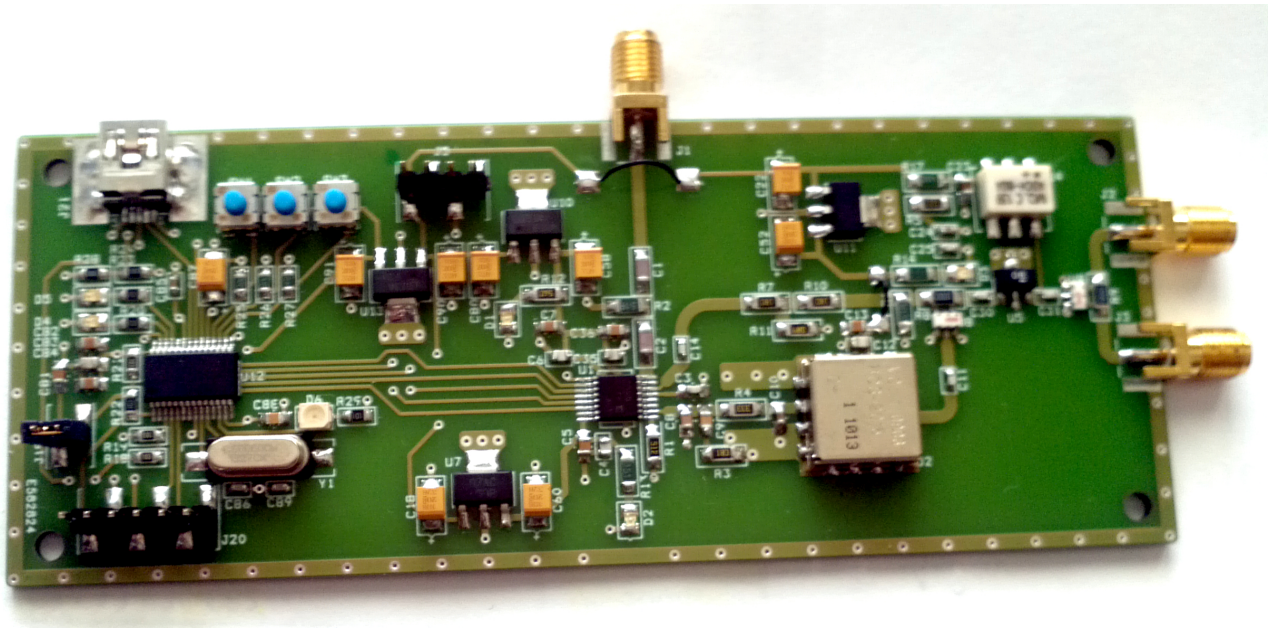


Figure A.4: Picture of the produced board with components soldered

Appendix B

PC Client User Manual

B.1 Introduction

This document details the installation process and explains how to use the PC client software for communication with the device.

B.2 Installation

The software is provided via an installer executable which copies all required files to a folder on the user's computer. At present the software is only provided for the Microsoft Windows operating system.

Compatible versions are:

- Microsoft Windows 7
- Microsoft Windows 8
- Microsoft Windows 8.1

When the installer wizard is executed, a welcome screen is presented (figure B.1) allowing the user to continue with the installation by pressing the "Next" button, or cancelling ("Cancel" button).

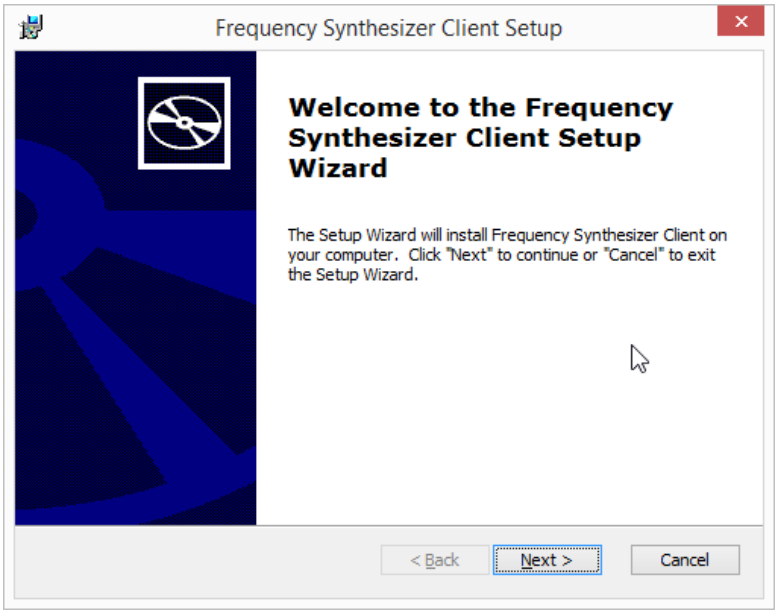


Figure B.1: Setup Wizard Welcome Screen

After pressing "Next" the user is presented with a folder Selection screen (figure B.2). This allows the user to change the folder where the software will be extracted to.

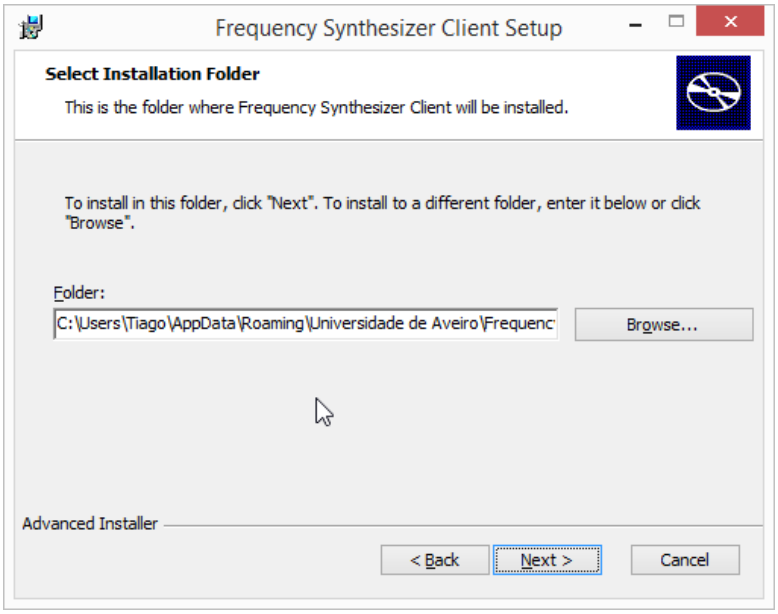


Figure B.2: Setup Wizard Folder Selection

When the folder is correct, the "Next" button takes the user to a confirmation screen (figure B.3), in which the user may select to proceed with the installation or cancel.

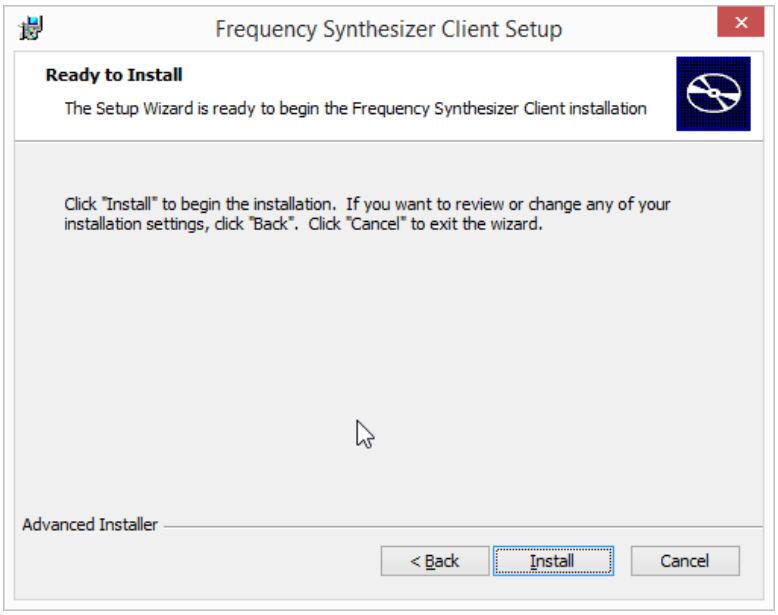


Figure B.3: Setup Wizard Confirmation

After installation is complete a finishing screen is shown (figure B.4). Here the user may choose to launch the installed program or to simply exit the installation.

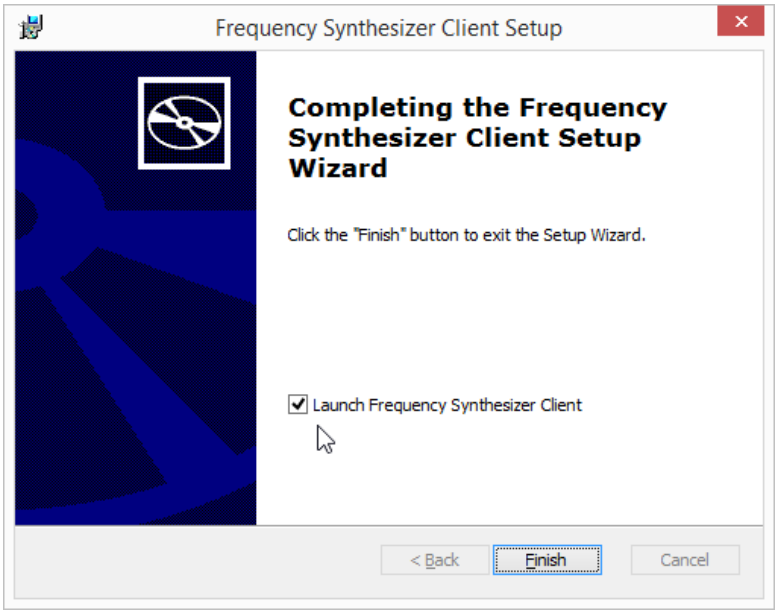


Figure B.4: Setup Wizard Installation Complete

Shortcuts are automatically placed in the user's Desktop as well as the start menu.

B.3 Usage

To open the program, the user simply needs to double-click the icon placed in the Desktop during installation. The program will function in any case but to program words to the device, the user must first connect the device before opening the program.

This program operates by storing and loading latch maps. Since the synthesizers used in the device are programmed using four different words which are combined to send all the needed information to the synthesizer, each frequency in the main window's frequency "spinner" (figure B.5), are calculated from a set of the four words needed to program the device. The frequency selector spinner has a limit of 16 words, which is the same as the hardware.

Selecting a frequency in the frequency selector will load the four word map associated with it, this will cause the "Latch Map Information" (figure B.6) area to update with the information of the words for the selected frequency. This area has information pertaining to the four words.

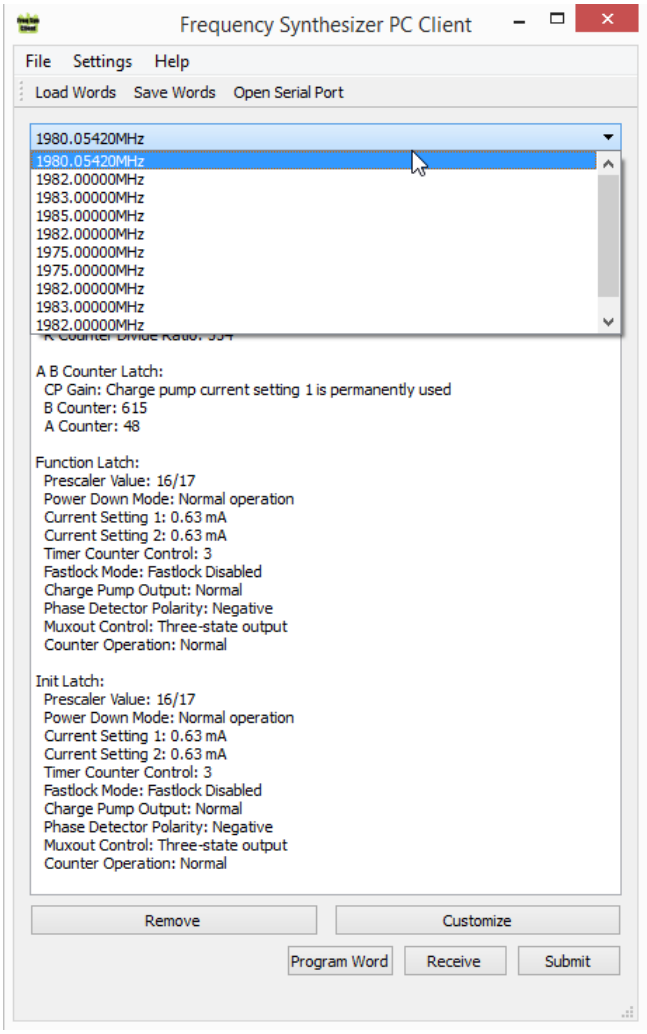


Figure B.5: Main Window Frequency Selector

The user may customize any word map by selecting it in the frequency selector and clicking the "Customize" button. This will take the user to the "customize" window (figure B.7). Here the user may individually adjust all the fields of the latch maps. The "Save" button will take the user's changes into account. If any of the counter values are altered by the user, the frequency in the title box will be recalculated.

The "Remove" button below the Latch Map Information text area in the main window will delete the selected frequency configuration.

The "Settings" menu in the main window has a single "Configuration" entry. This is where

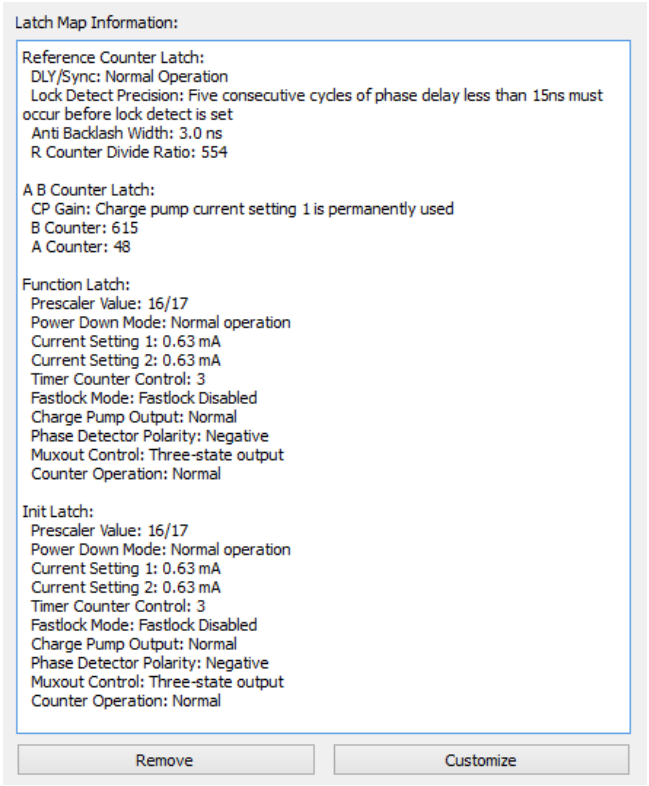


Figure B.6: Main Window Latch Information

---

the user can change all the settings relating not only to the program but to the device as well. The Settings window (figure B.8) is divided in four sections. The top two sections pertain to the Communication parameters. The first section in the top left of the window has a selector for the user to change the serial port connected to the device. The top right section ("Parameters") allows the user to change the comm settings for the serial communication. The values configured in the device are the default ones, but the user may change them how he pleases.

The center section ("Additional Options") allows the user to change the serial echo option (mainly for debugging purposes), and to select a different word file, that is, the file where all frequency maps and word combinations are stored.

The bottom section has values dependent on the device. The values stored are loaded from an external "settings" file. When a new device is created using a different synthesizer chip or a different VCO, these values may be changed, the settings file may then be supplied to all the users who use the new version of the hardware. This applies also to when the user wants

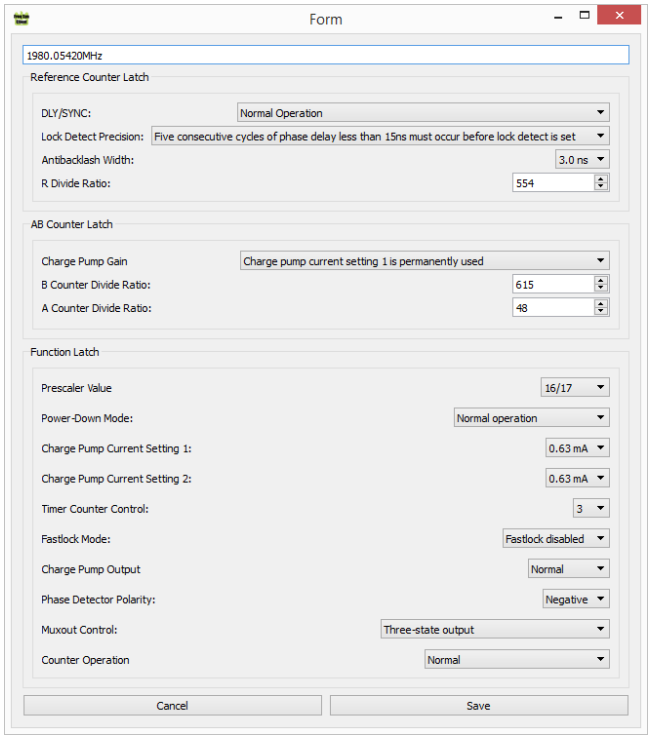


Figure B.7: Customize Window

to test word combinations which will create frequencies higher or lower than the ones allowed. The R Counter and prescaler values are also dependent on the hardware but in this case they relate to the loop filter, which in turn is dependent on the intended frequency range.

B.3.1 Programming the words

There are three buttons near the bottom of the main window. The word map selected in the frequency selector is the one operated upon by these buttons. That is if the "Program Word" button is clicked, the selected word is sent to the device's storage and immediately programmed. If the "Submit" button is pressed, the selected word is sent to the corresponding index in the device. This means that even if this word is word number 16 (maximum amount of words) in the PC client, and there are no words programmed in the device, this word will have index 16 in the device.

This allows the user to program all the words in sequence or not, but not having to remember their position on the device's memory. The PC Client will have all the words in the desired

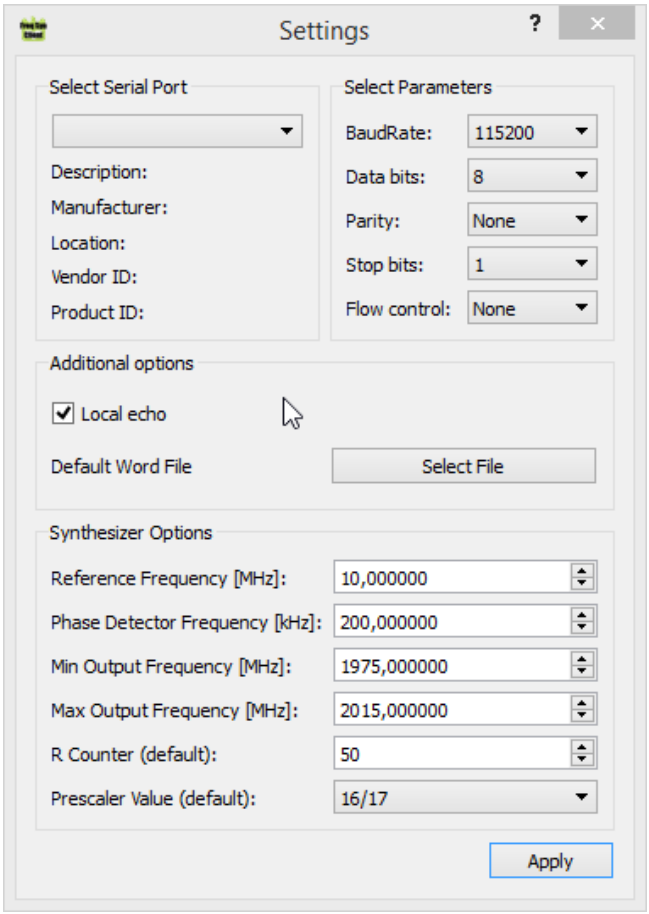


Figure B.8: Settings Window

order and they can be programmed to the device one at a time.

As the name says, the "Receive" button will substitute the selected word by the one of the same index in the device (will only work if the device has a word map in that index).

Appendix C

Devices for Different Output Frequencies

C.1 330 to 540 MHz

Utilizing Analog Device's ADF4110 synthesizer and Mini-Circuit's ROS-EDR4828 VCO, we can produce a synthesizer with output frequencies between $330Mhz$ and $540MHz$. Using a phase detector frequency (channel spacing) of $50kHz$ and a reference of $104MHz$, we have the result that may be seen in C.1

C.2 610 to 1120 MHz

Utilizing Analog Device's ADF4111 synthesizer and Mini-Circuit's ROS-1120-119+ VCO, we can produce a synthesizer with output frequencies between $610Mhz$ and $1120MHz$. Using a phase detector frequency (channel spacing) of $100kHz$ and a reference of $104MHz$, we have the result that may be seen in C.2

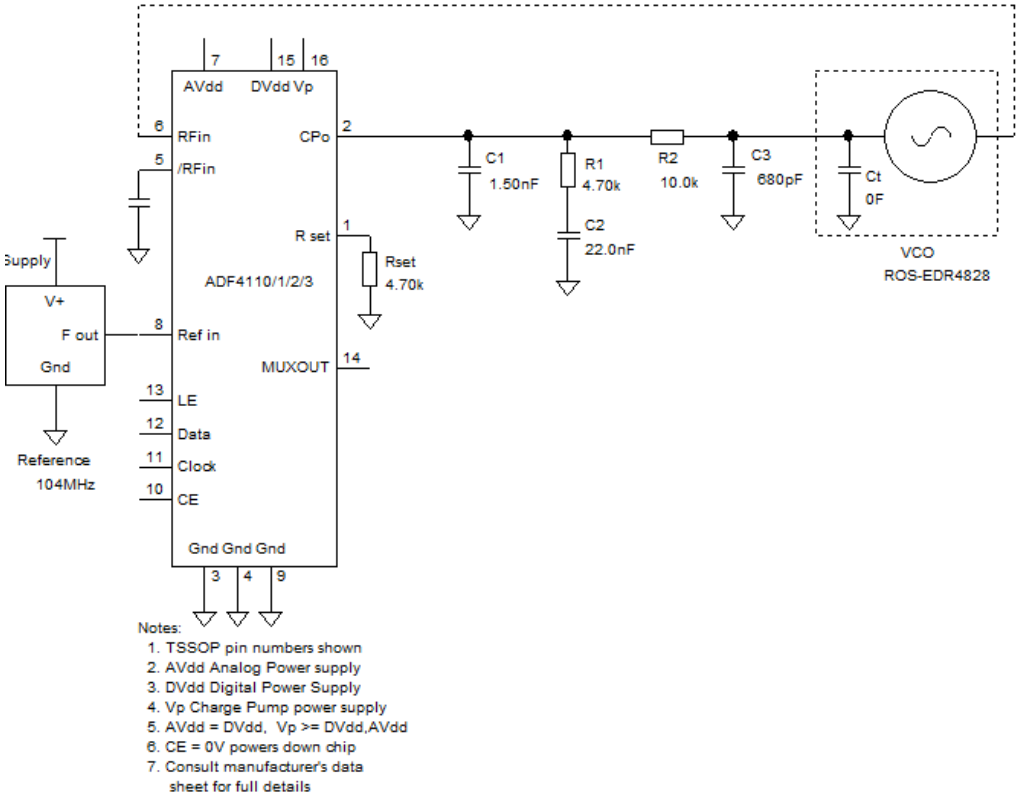


Figure C.1: Loop Filter Schematic - 330 to 540Mhz

C.3 3000 to 3500 MHz

Utilizing Analog Device's ADF4113 synthesizer and Mini-Circuit's ROS-3600-619+ VCO, we can produce a synthesizer with output frequencies between $3000Mhz$ and $3500MHz$. Using a phase detector frequency (channel spacing) of $250kHz$ and a reference of $104MHz$, we have the result that may be seen in C.3

All of the calculated frequency options have similar phase noise predictions in the calculation software, which can be seen in image C.4

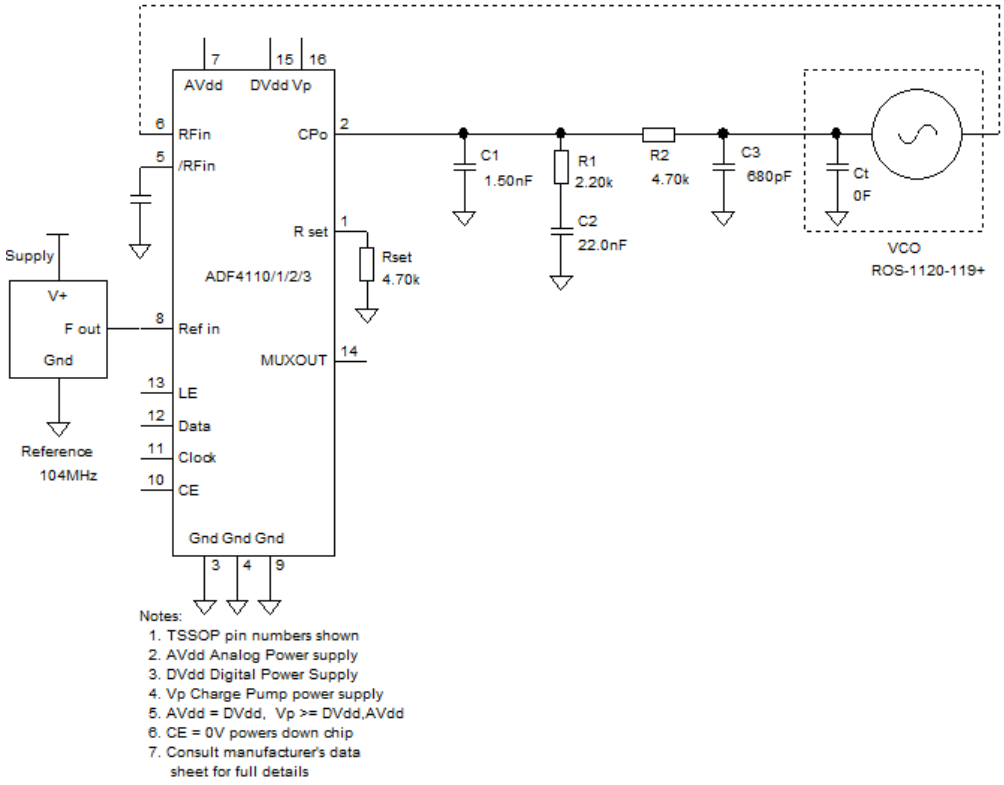


Figure C.2: Loop Filter Schematic - 610 to 1120 MHz

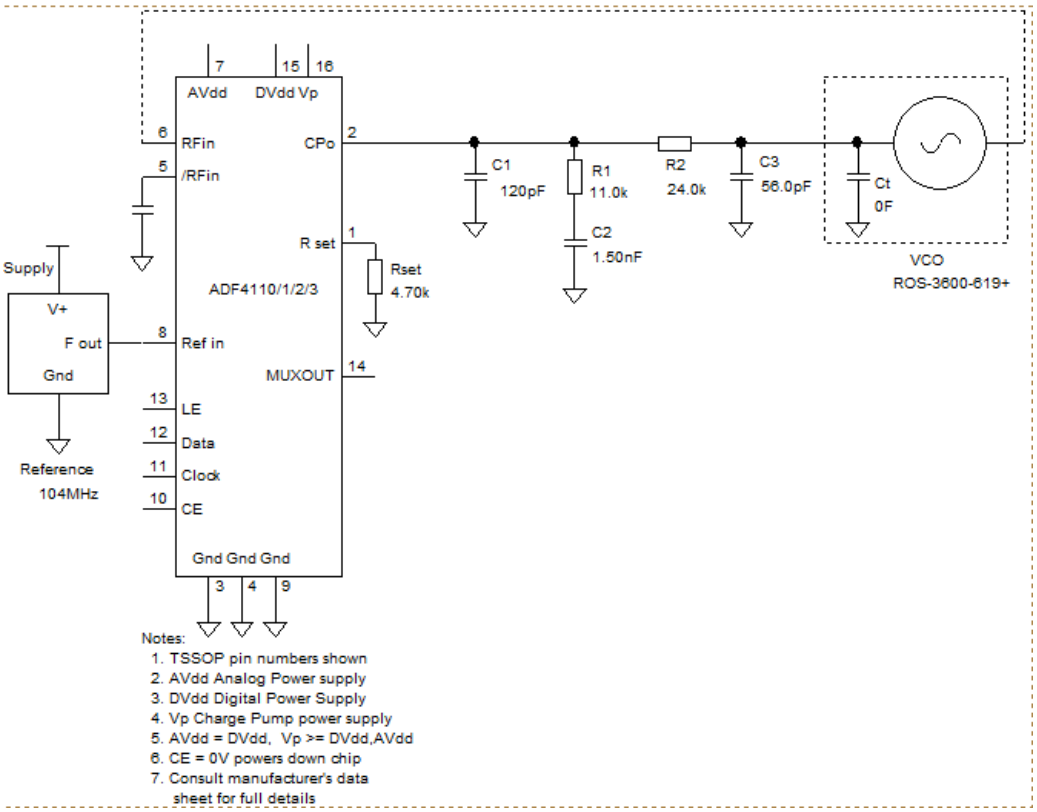


Figure C.3: Loop Filter Schematic - 3000 to 3500 MHz

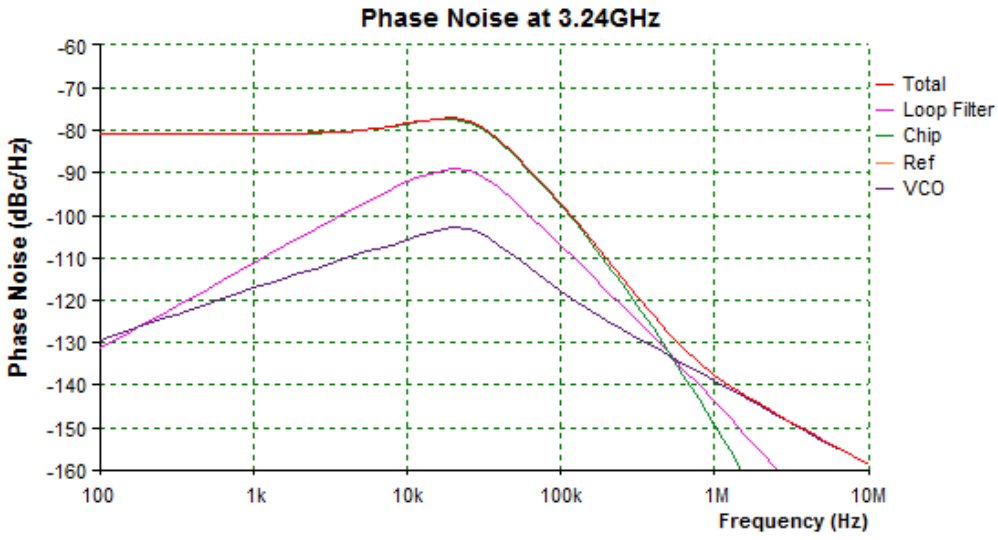


Figure C.4: Phase Noise calculated by ADISimPLL